MASTER EN INGÉNIEUR DE GESTION À FINALITÉ SPÉCIALISÉE EN DATA SCIENCE

Modification conviviale d'arbres de décision par l'utilisateur via l'interaction avec des règles décisionnelles induites

Bernard, François

Award date:
2021

Awarding institution:
Universite de Namur

Modification conviviale d'arbres de décision par l'utilisateur via l'interaction avec des règles décisionnelles induites

Bernard François

Directeur : Prof B. Frénay

Co-directeur : G. Nanfack

Mémoire présenté

en vue de l'obtention du titre de

Master 120 en ingénieur de gestion, à finalité spécialisée

en data science

ANNEE ACADEMIQUE 2020-2021

Table des matières

Chapitre 1

Introduction

L'intelligence artificielle (IA) est une des plus grandes inventions du siècle dernier. Son origine remonte à la création du test de Turing en 1950. A ses côtés se développe le *machine learning*, terme utilisé pour la première fois en 1959 par le mathématicien Arthur Samuel (Samuel 1959). Il s'est penché sur le jeu de dames et son programme a battu le 4ème joueur des Etats-Unis en 1961. Après un engouement certain dans les années 1960 et 1970, l'intelligence artificielle connaîtra deux déclins nommés *hivers de l'IA*. Cependant, depuis les années 1990 et l'amélioration de la puissance de calcul des ordinateurs, l'intelligence artificielle et le machine learning connaissent un envol significatif. De nos jours, l'IA est bien ancrée dans notre société. Elle est autant utile pour les réseaux sociaux que pour la lutte contre le réchauffement climatique ou la construction automobile.

Le machine learning est une branche l'intelligence artificielle. Il se divise en trois grandes catégories : l'apprentissage supervisé, l'apprentissage non supervisé et l'apprentissage par renforcement. L'apprentissage supervisé a principalement un but prédictif. Cela signifie qu'à partir de données dont on connaît la caractéristique cible, l'objectif est de pouvoir estimer la valeur de cette caractéristique cible pour des nouvelles données inconnues. Un exemple concret pourrait se prêter à l'estimation du prix d'une maison. Premièrement, il est nécessaire d'avoir à notre disposition un dataset contenant des centaines, voire des milliers de maisons. Pour chacune de ces maisons, plusieurs caractéristiques influant sur le prix sont connues, telles que la superficie, le nombre de chambres, la superficie du jardin, la situation géographique, etc. Dans le cas de l'apprentissage supervisé, le prix des maisons est également connu. Ensuite, toutes les données sont passées à travers un algorithme de machine learning. Celui-ci tire des conclusions et est capable d'estimer le prix de maisons dont il connaît les caractéristiques. L'apprentissage non supervisé a, lui, un but de reconnaissance de motifs dans le

dataset. Dans notre exemple de dataset de maisons, il n'y aurait plus le prix des maisons. Les algorithmes de machine learning auraient pour objectif de regrouper les différentes maisons en groupes de maisons similaires au niveau de leurs caractéristiques.

Bien que le machine learning soit extrêmement utile dans la société actuelle, il existe plusieurs problématiques auxquelles il doit faire face. En effet, le machine learning se base sur un ou plusieurs *datasets* composés d'instances. A travers de multiples algorithmes, le machine learning va tenter d'extraire de la connaissance à partir de ces datasets. Cependant, si ces derniers sont de mauvaise qualité, les conclusions tirées par les algorithmes de machine learning peuvent être erronées. Dans un dataset, il peut y avoir des erreurs, des valeurs manquantes ou des biais. Une expression populaire exprimant cette problématique est *garbage in, garbage out* qui signifie que si un modèle apprend à partir de données erronées, il pourrait en sortir des conclusions erronées.

Pour pallier ce problème de qualité dans les données, énormément d'experts se sont penchés sur de possibles solutions. Encore aujourd'hui, les données manquantes, erronées ou encore les biais dans les datasets sont des freins restreignant la puissance du machine learning. Cet essai présente donc une approche permettant à un expert du domaine de recherche d'apporter son expérience et ses connaissances pour *améliorer* un modèle de machine learning préalablement entraîné.

Concrètement, la méthodologie que nous allons présenter dans le chapitre 5 se déroule en trois grandes étapes. La première étape consiste en l'induction d'un ensemble de règles décisionnelles. La deuxième étape comprend la modification des règles par un ou plusieurs experts. Enfin, lors de la troisième étape, l'ensemble de règles modifié est transformé en un arbre de décision. Pour présenter cette méthodologie de la manière la plus compréhensible possible, cet essai introduit les concepts d'arbres de décision et les ensembles de règles de décision dans le chapitre 2. Une revue de la littérature suit dans le chapitre 3. Elle concerne les différents algorithmes qui permettent de générer un ensemble de règles à partir d'un dataset et les algorithmes qui génèrent un arbre de décision à partir de règles de décision. Une analyse des différentes interviews réalisées suit dans le chapitre 4. La méthodologie en tant que telle est introduite et développée dans le chapitre 5. Ensuite, le logiciel développé pour implémenter la méthodologie est présenté dans le chapitre 6. Une expérimentation de l'approche sur le logiciel en question suit dans le chapitre 7. Le chapitre 8 développe des pistes pour l'implémentation future de contraintes haut niveau. Enfin, le chapitre 9

conclut cet essai avec les différents résultats obtenus et les améliorations possibles dans le futur.

Chapitre 2

Arbres de décision et règles de décision

Avant de commencer à rentrer dans l'explication de la méthodologie en tant que telle, il est important d'expliquer les différents concepts de machine learning nécessaires à la compréhension de la méthodologie. Premièrement, la notion de dataset et sa formalisation mathématique sont présentées. Ensuite, dans la section 2.2, les arbres de décision sont présentés tandis que dans la section 2.3, les règles de décision sont abordées.

2.1 Dataset

Pour pouvoir aborder les arbres de décision et les règles de décision dans la plus grande clarté, il convient de jeter un coup d'oeil plus attentif à ce qu'est mathématiquement un dataset. Comme introduit dans le livre (Bonaccorso 2017), un dataset D est un "ensemble fini de vecteurs avec d + 1 caractéristiques chaque". Ce dataset est formulé mathématiquement dans ce livre par $X = X_1, X_2, ..., X_n$ avec $X_i \in \mathbb{R}^{d+1}$. Chaque *vecteur* peut également être appelé une *instance*. La i-ème instance sera notée X_i tandis que la valeur du j-ème attribut de cette instance sera notée X_{ij}. Le j-ème attribut sera quant à lui référencé par A_j. Lorsqu'il est question d'apprentissage supervisé, chaque instance possède également une caractéristique cible qui peut être appelée *classe*. La classe de l'instance X_i sera notée t_i. Il est donc possible de représenter un dataset par une matrice de données (voir figure 2.1a) et un vecteur cible (voir figure 2.1b).

Notons que dans la méthodologie présentée dans la section 5, seuls des attributs catégoriels sont utilisés. Il existe donc un nombre fini de valeurs présentes pour chaque attribut dans le dataset.

$$\mathbf{X} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{pmatrix}$$

(a) Matrice de données d'un dataset, reprise de (Frénay 2020)

$$\mathbf{t} = \begin{pmatrix} t_1 \\ t_2 \\ \vdots \\ t_n \end{pmatrix}$$

(b) Vecteur cible d'un dataset, repris de (Frénay 2020)

L'ensemble de toutes les valeurs possibles de l'attribut A_i est noté S_i. L'ensemble de toutes les classes présentes dans le dataset est quant à lui noté C.

2.2 Arbres de décision

Un arbre de décision se présente sous la forme de noeuds, de feuilles et de branches. Sur la figure 2.2, les noeuds sont représentés par des rectangles, les feuilles par les mots *No* et *Yes* et les branches par des lignes. Comme décrit dans (Mitchell et al. 1997), un arbre de décision classifie les instances en partant du dessus de l'arbre et en descendant jusqu'à une des feuilles.

2.2.1 Arbres binaires et non-binaires

Pour pouvoir définir précisément tous les éléments d'un arbre de décision, les deux types d'arbres de décision doivent être présentés : les arbres binaires et les arbres non-binaires. Lorsqu'il est question d'un arbre non-binaire, chaque noeud n'a pas de limite de branches tandis que dans un arbre binaire tous les noeuds possèdent exactement deux branches. Cette distinction a un impact lorsqu'il s'agit de définir un noeud. En effet, un noeud dans un arbre non-binaire spécifie simplement un attribut et chacune des branches découlant de celui-ci fait référence à une des valeurs de l'attribut (voir figure 2.3a). Dans un arbre binaire, chaque noeud correspond à une condition sous la forme $A_i \in V_i$ avec A_i un attribut et V_i un ensemble de valeurs pour l'attribut en question. Dans ce cas, une des deux branches découlant d'un noeud représente l'affirmation de la condition tandis que l'autre correspond à la négation de cette condition (voir figure 2.3b). Dans le reste de cet essai, seulement les arbres binaires sont abordés.

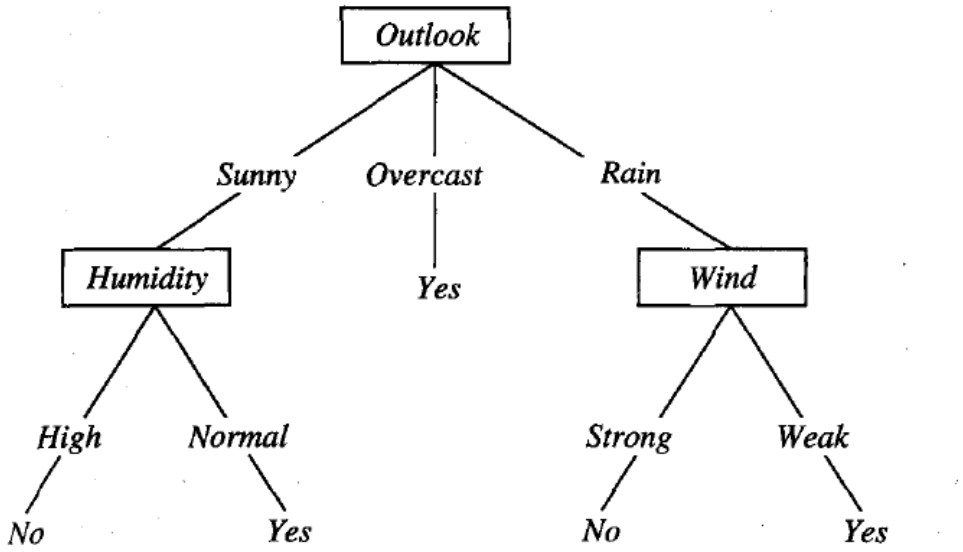
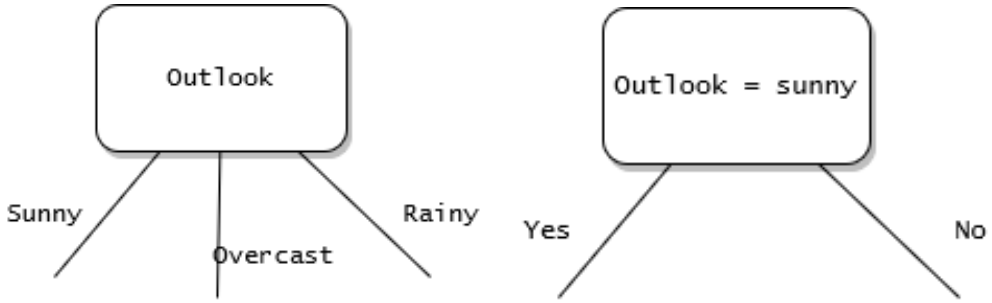


Figure 2.2: Exemple d'arbre de décision, tiré à partir de (Mitchell et al. 1997)



(a) Exemple de noeud non-binaire (b) Exemple de noeud binaire

Figure 2.3: Exemples de noeuds binaires et non-binaires

2.2.2 Classification d'une instance

Pour classifier une instance, il faut descendre l'arbre de noeud en noeud jusqu'à atteindre une feuille. Pour cela, la condition inscrite sur le noeud est analysée pour l'instance en question. En fonction de la valeur de l'attribut pour l'instance, la branche correspondante sera suivie. Lorsque l'instance atteint une feuille, il est possible de la classifier en fonction de la classe indiquée sur la feuille qu'elle a atteint. L'exemple proposé sur la figure 2.2 et tiré de (Mitchell et al. 1997), propose de prédire si un dimanche est un jour *convenable* pour jouer au tennis. Tout d'abord, l'arbre de décision nous indique de vérifier si le temps est ensoleillé (sunny), nuageux (overcast) ou pluvieux (rainy). S'il est ensoleillé, alors le niveau d'humidité est analysé. Un haut niveau d'humidité ne conviendrait pas à la pratique du tennis tandis qu'un taux d'humidité normal serait parfait. Si le temps est nuageux,

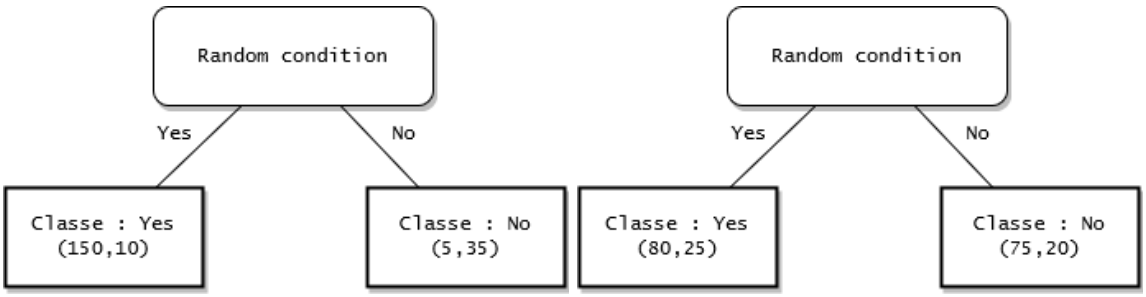
l'instance arrive directement à une feuille contenant la classe *Yes* ce qui signifie que le jour est idéal pour pratiquer le tennis. Enfin, si le temps est pluvieux, l'attribut *force du vent (wind)* sera décisif. Si le vent est fort, les raquettes ne seront pas de sortie tandis qu'un vent faible permettra de jouer malgré la pluie.

2.2.3 Formulation mathématique d'un arbre de décision

Il est désormais possible de décortiquer plus en profondeur les différents éléments constituant un arbre de décision. Un noeud va comporter une condition (dans un arbre binaire) que l'on peut généraliser par $A_i \in V_i$ avec V_i un ensemble de valeurs possibles pour l'attribut A_i. Cette condition peut également être notée sous la forme $A_i = V_{ij}$ ou encore $A_i \leq V_{ij}$ dans le cas d'attributs non catégoriels. Une feuille quant à elle ne contient pas de condition mais une classe. Cette classe est la prédiction du modèle pour toutes les instances qui arriveraient jusqu'à cette feuille. Chaque noeud et chaque feuille possède également un ensemble d'instances qui lui *appartiennent*. Pour déterminer si une instance appartient à un noeud ou à une feuille, il suffit de réaliser le chemin de l'instance à travers l'arbre tout en observant si l'instance passe par le noeud ou la feuille en question. Pour qu'une instance appartienne à un noeud, toutes les conditions indiquées sur les noeuds des niveaux supérieurs doivent être respectées par l'instance.

Pour chaque noeud et chaque feuille, il est dès lors possible d'obtenir la répartition dans les différentes classes des instances du dataset. Sachant que le but principal de l'arbre de décision est de classifier correctement les futures instances, il est intéressant que la répartition en classes des différentes feuilles soit la plus séparée possible. Si toutes les instances appartenant à une feuille proviennent à 50% de la première classe et à 50% de la seconde, il est difficile d'attribuer une classe à cette feuille. De même, plus les instances se rapprochent des 100% d'appartenance à une seule classe, plus l'algorithme est *confiant* dans sa prédiction. Pour illustrer ceci avec un exemple, un noeud est créé dont 155 de ses instances appartiennent à la classe Yes et les 45 autres à la classe No. La figure 2.4a montre une condition qui sépare bien les instances des deux classes. En effet, 96.77% des instances de la classe Yes suivront la branche de gauche tandis et 77.78% des instances de la classe No suivront la branche de droite. Le modèle prédit avec beaucoup de certitude la classe Yes pour toutes les futures instances qui atteignent la feuille de gauche et la classe No pour toutes celles qui appartiennent à la feuille de droite. Sur la figure 2.4b cependant, les instances sont nettement moins bien séparées. Environ la moitié des instances de chaque classe se retrouve dans

chacune des deux feuilles. Le modèle est bien moins convaincu et convaincant dans sa prédiction avec cette séparation. La plupart du temps, contrairement à l'exemple de la figure 2.4b, la classe



(a) exemple de bonne séparation des instances (b) exemple de mauvaise séparation des instances

Figure 2.4: Exemples de séparation des instances

suggérée par le modèle pour une feuille correspond à la classe majoritaire dans les instances du dataset d'entraînement appartenant à cette feuille.

Sans rentrer dans les détails techniques du fonctionnement d'un algorithme d'induction d'arbres de décision, le modèle va tenter de déterminer la condition optimale pour séparer les instances en deux branches distinctes. Pour réaliser cela, il utilise le principe de séparation des classes abordé dans le paragraphe précédent. Plus les groupes d'instances formés par une condition sont *homogènes*, plus la condition a de chance de figurer sur le prochain noeud.

La grande force d'un arbre de décision, s'il ne contient pas trop de branches et de noeuds, est sa lisibilité et son interprétabilité. Il est en effet assez aisé de comprendre le processus prévisionnel sous-jacent. Les algorithmes générant des arbres de décision les plus connus sont ID3 (Quinlan 1983) et C4.5 (Quinlan 2014).

2.3 Règles de décision

Pour présenter les règles de décision, il faut faire appel à la logique du première ordre comme expliqué dans (Mitchell et al. 1997). Une règle de décision se présente sous une forme *IF-THEN* comme sur l'exemple suivant.

$$\text{IF } outlook = (sunny, overcast) \wedge wind = (weak) \text{ THEN } play = yes$$

Une règle de décision est donc une conjonction de conditions où une condition est une liste de valeurs possibles pour un attribut. Une règle peut donc s'écrire :

$$R_j \equiv \text{IF } c_1 \wedge c_2 \wedge ... \wedge c_k \text{ THEN } C_i$$

où $c_i \equiv A_i \in V_i$ avec i l'index de l'attribut en question A_i et V_i l'ensemble des valeurs possibles pour l'attribut dans la condition c_i et C_i la classe prédite par la règle. Si un attribut n'est pas indiqué dans une règle de décision, cela signifie que sa valeur n'importe pas dans la règle. Il est alors possible de simplement noter : $c_i \equiv A_i \in S_i$ avec S_i l'ensemble contenant toutes les valeurs possibles pour l'attribut A_i.

Une règle de décision n'est cependant pas suffisante pour prédire efficacement la classe des instances. En effet, si le modèle généré se limite à une seule règle de décision, il est impossible de prédire la classe de toutes les instances n'appartenant pas à cette règle. Un ensemble de règles ne contenant qu'une seule règle est comparable à un arbre à une seule branche et donc une seule feuille. Il est donc nécessaire d'induire plusieurs règles qui formeront un ensemble de règles de décisions (ou *ruleset* en anglais).

Pour déterminer la classe prédite par le modèle pour une instance, il faut déterminer à quelle règle elle *appartient*. Ce processus est similaire à celui expliqué dans la section 2.2 pour définir l'appartenance d'une instance à une feuille de l'arbre. Pour cela, il faut vérifier si l'instance correspond à chacune des conditions de la règle. Par exemple, la quatrième instance du tableau 2.5 appartient à la règle formulée ci-dessus tandis que la deuxième instance n'appartient pas à la règle car l'attribut *outlook* est égal à *rainy* et la condition $weather = (sunny, cloudy)$ n'est pas respectée. On formalise cette relation d'appartenance pour l'instance X_i à la règle R par la formule :

$$Appartenance(R, X_i) \equiv \forall j \leq d : X_{ij} \in V_j \tag{2.1}$$

avec d le nombre d'attributs des instances, X_{ij} la valeur de l'attribut en question et V_j l'ensemble de valeurs possibles de l'attribut dans la condition c_j de la règle R.

Outlook	Humidity	Wind
Sunny	High	Strong
Rainy	Normal	Weak
Overcast	Normal	Strong
Sunny	Normal	Weak

Figure 2.5: Exemple de dataset réduit

2.4 Propriétés d'un ensemble de règles de décision

Un ensemble de règles de décision peut avoir deux propriétés principales : la complétude et la disjointeté. Pour qu'un ensemble de règles soit complet, chaque instance possible dans l'espace de possibilité des valeurs de ses attributs doit être couverte par au moins une règle de décision. Pour qu'un ensemble de règles de décision soit disjoint, aucune instance ne doit être couverte par plus d'une règle.

La disjointeté entre deux règles R_1 et R_2 est déterminée par la fonction

$$Disjointness(R_1, R_2) \equiv \begin{cases} True, & \text{si } \exists i \leq d : V_{i1} \neq V_{i2} \wedge V_{i1} \neq S_i \neq V_{i2} \\ False, & \text{sinon} \end{cases} \quad (2.2)$$

avec d le nombre d'attributs, V_{ij} l'ensemble des valeurs pour l'attribut A_i dans la règle R_j et S_i l'ensemble de toutes les valeurs possibles de cet attribut. La disjointeté d'un ensemble de règle peut ensuite être évalué par la fonction

$$DisjointessRuleset(Ruleset) \equiv \begin{cases} True, & \text{si } \forall i, j \leq k : i \neq j \wedge Disjointness(R_i, R_j) \\ False, & \text{sinon} \end{cases} \quad (2.3)$$

avec k le nombres de règles dans l'ensemble de règles.

En prenant comme exemples les 3 règles suivantes du tableau 2.2, les règles R_1 et R_2 sont disjointes, comme les règles R_2 et R_3. Toutefois, les règles R_1 et R_3 ne sont pas disjointes. Cela a pour

Règle 1	R_1 : IF $A = 1 \wedge B = 1$ THEN Yes
Règle 2	IF R_2 : IF $A = 2 \wedge C = 1$ THEN No
Règle 3	R_1 : IF $A = 1 \wedge C = 1$ THEN No

Table 2.1: Exemple de ruleset

conséquence qu'un individu dont les attributs sont $A = 1 \wedge B = 1 \wedge C = 1$ appartiendrait aux 2 règles.

La complétude d'un ruleset peut, elle, se définir par la fonction

$$Complete(Ruleset) \equiv \begin{cases} True & \text{si } \forall X : \exists R : Appartenance(R, X) \\ False & \text{sinon} \end{cases} \quad (2.4)$$

avec X n'importe quelle instance connue ou inconnue et R une règle appartenant à $Ruleset$.

Un ensemble de règles non-disjointes peut poser un problème de prédiction dans le cas suivant. Si une instance dans le dataset est couverte par plusieurs règles dont la classe prédite est différente, il est nécessaire de déterminer quelle règle prime sur l'autre et donc quelle classe prédire pour l'instance en question. Un ensemble de règles non-complet pose un autre type de problématique. En effet, quelle classe attribuer à une instance n'appartenant à aucune règle de l'ensemble ? Ces deux problématiques sont abordées dans la section 5.

2.5 Relation entre un arbre de décision et un ensemble de règles

Un arbre de décision et un ensemble de règles sont intimement liés par le fait qu'un arbre de décision est un type spécial d'ensemble de règles de décision. En effet, chaque *chemin décisionnel* possible du sommet de l'arbre jusqu'à une feuille est une règle de décision dont chaque noeud est une condition. Par exemple, dans l'arbre présenté à la figure 2.2, la branche la plus à gauche correspond à la règle décisionnelle

$$\text{IF } outlook = sunny \wedge humidity = high \text{ THEN } No.$$

En regroupant tous les chemins décisionnels possibles de l'arbre de décision, on obtient donc un ensemble de règles. Pour le même arbre de décision on obtient donc l'ensemble de règles présent au tableau 2.2. En allant plus loin dans l'analyse, il est intéressant de remarquer que cet ensemble de règle est disjoint et complet. Dans un arbre non-binaire, toutes les valeurs possibles de l'attribut du noeud sont représentés par une branche. Une fois encore, il est toujours possible de déterminer

Règle 1	IF $outlook = sunny \wedge humidity = high$ THEN No
Règle 2	IF $outlook = sunny \wedge humidity = normal$ THEN Yes
Règle 3	IF $outlook = overcast$ THEN Yes
Règle 4	IF $outlook = rain \wedge wind = strong$ THEN No
Règle 5	IF $outlook = rain \wedge wind = weak$ THEN Yes

Table 2.2: Ruleset tiré de l'arbre de décision

la branche à suivre pour l'instance. Changer la forme de notre modèle en passant d'un arbre de décision à un ensemble de règles décisionnelles est donc une opération assez simple à réaliser. Toutefois, l'opération inverse se révèle beaucoup plus complexe et plus d'explications sont données dans la section 5.

2.6 Conclusion

Dans ce chapitre les notions de dataset, d'arbre de décision et de règle de décision sont présentées. Deux propriétés intéressantes d'un ensemble de règles de décision sont également mentionnées : la *complétude* et la *disjointeté*. Enfin, un arbre de décision est un type spécifique d'ensemble de règles, complet et disjoint comme expliqué dans la section 2.5. Cette présentation des concepts a pour but de simplifier la compréhension future des algorithmes observés dans le chapitre 3 et de la méthodologie développée dans le chapitre 5.

Chapitre 3

État de l'art de la génération de règles et d'arbres de décision

Dans cette section, plusieurs algorithmes de la littérature sont analysés. deux types d'algorithmes sont présents : les algorithmes induisant des règles de décision et les algorithmes induisant des arbres de décision.

3.1 Algorithmes d'induction de règles de décision

Beaucoup de chercheurs se sont penchés sur la problématique de l'induction de règles décisionnelles. Lorsque l'on tente d'induire un ensemble de règles de décision, deux objectifs important principalement et il convient de les aborder conjointement.

Le premier objectif concerne la qualité de l'ensemble de règles au niveau prédictif. Pour mesurer cette qualité prédictive, il existe plusieurs métriques. Un premier indicateur est le pourcentage d'instances correctement classifiées. Un modèle dont le pourcentage d'instances correctement classifiées est maximal (ou inversement, dont le pourcentage d'erreur de classification est minimal) serait donc privilégié. Cependant, il est parfois préférable de se tromper dans la classification d'une classe plutôt que dans les autres. Par exemple, un médecin préfère identifier un patient comme malade même s'il ne l'est pas plutôt que d'identifier un patient comme sain alors qu'il est malade. Dans ces situations, observer une matrice de confusion plutôt que le pourcentage d'instances correctement classifiées peut être pertinent. Sur la figure 3.1, une matrice de confusion fictive montre que 89% des instances sont correctement classifiées. Toutefois, sur les 190 cas positifs, seulement 90 sont classifiés

Confusion Matrix		
	Actual : Positive	Actual : Negative
Predicted : Positive	90	10
Predicted : Negative	100	800

Figure 3.1: Exemple de matrice de confusion

correctement. Cela représente un pourcentage de 10% de faux négatifs et 1% de faux positifs. Ce genre de modèle ne serait donc pas privilégié dans des domaines où les faux négatifs sont à proscrire.

Le deuxième objectif d'un algorithme d'induction de règles de décision est de simplifier au maximum l'ensemble de règles. En effet, un trop grand nombre de règles de décision dans un ensemble crée un problème appelé *overfitting*. L'overfitting est défini comme ceci dans (Mitchell et al. 1997): "Donné un espace d'hypothèses H, une hypothèse $h \in H$ *surapprend (overfit)* les données d'entraînement si il existe une hypothèse alternative $h' \in H$ tel que h a une erreur plus faible que h' sur les données d'entraînement, mais que h' a une erreur plus faible que h sur l'ensemble des données."

Ce problème survient donc lorsqu'un modèle de machine learning devient trop complexe et *colle* presque parfaitement au dataset de départ. A première vue, ce modèle est très performant car il atteint un score de presque 100% d'instances correctement classifiées dans le dataset d'entraînement. Cependant ce genre de modèle est peu performant sur de nouveaux datasets. Dans le cas concret des ensembles de règles décisionnelles, créer autant de règles qu'il y a d'instances permettrait d'obtenir un score de 100% lors de la classification des instances du dataset. Cependant, cela poserait une problème d'overfitting et le score du modèle sur de nouvelles instances serait faible.

Cette réduction de la taille de l'ensemble de règles peut être réalisée en supprimant certaines règles

de l'ensemble ou en supprimant certaines conditions dans les règles. Deux méthodes principales sont apparues dans la littérature : le *pre-pruning* et le *post-pruning*. Ces deux méthodes permettant de contraindre l'overfitting et son opposé l'underfitting sont expliqués plus en détail dans la section suivante.

3.1.1 IREP

Avant de présenter l'algorithme IREP comme décrit dans (Fürnkranz and Widmer 1994), les notions de *pruning*, *pre-pruning* et *post-pruning* sont introduites pour faciliter la compréhension.

Le pruning est décrit dans (Mingers 1989) comme un moyen de gérer le bruit dans les données. Les méthodes de pruning ont donc pour but de produire un modèle plus général pour éviter les problèmes d'overfitting sur les instances encore inconnues. Le pruning se divise en deux grandes méthodes différentes : le pre-pruning et le post-pruning. Dans (Fürnkranz and Widmer 1994), le pre-pruning est défini de la manière suivante : "Le pre-pruning signifie que pendant la génération de concept, plusieurs exemples d'entraînement sont ignorés, de manière à ce que la description du concept ne classifie pas toutes les instances d'entraînement correctement.". Dans le même article, le post-pruning est caractérisé par le fait qu'il "signifie qu'une description du concept expliquant parfaitement toutes les instances d'entraînement est d'abord générée. Cette théorie est ensuite généralisée en coupant certaines branches de l'arbre de décision". Dans le pre-pruning, la généralisation du modèle se déroule donc pendant l'entraînement du modèle tandis que cette généralisation apparaît après l'entraînement pour une méthode de post-pruning.

IREP ou *Incremental Reduced Error Pruning* est une évolution de l'algorithme REP ou *Reduced Error Pruning* développé par Brunk and Pazzani (Brunk and Pazzani 1991). Ce dernier utilise le post-pruning pour généraliser un modèle qui classifie correctement toutes les instances du training set (appelé dans ce cas-ci *growing set*). Ce post-pruning est réalisé en retirant des règles ou des conditions de la théorie jusqu'à ce qu'une future suppression engendre une baisse de la performance du modèle sur le test set (appelé *pruning set*).

Malgré l'efficacité de REP avec des datasets contenant du bruit, cette méthode a plusieurs limites. Dans (Fürnkranz and Widmer 1994), les auteurs recensent quatre problèmes principaux de la méthode REP. Premièrement, la complexité de l'algorithme est égale à $\Omega(n^4)$ avec n le nombre

d'instances, comme prouvé dans (William W. Cohen 1993). Cette complexité est très élevée et rend l'algorithme assez lent quand le nombre d'instances augmente. Un deuxième problème réside dans la séparation entre le *growing set* et le *pruning set*. Avec REP, 2/3 du dataset est utilisé pour composer le growing set tandis que le tiers restant représente le pruning set. Si cette séparation du dataset est inappropriée, certaines règles importantes pourraient être oubliées lors de la phase d'apprentissage et d'autres règles correctement apprises pourraient être supprimées lors de la phase de pruning. Troisièmement, REP utilise un algorithme *separate-and-conquer* (Bagallo and Haussler 1990). Avec cette méthode, toutes les instances couvertes par une règle sont supprimées pour l'apprentissage des règles restantes. Cependant, étant donné que REP n'utilise pas de pre-pruning lors de cette phase d'apprentissage, chaque règle n'est pas généralisée et donc un nombre plus faible d'instances sont couvertes par une règle. La conséquence est un ensemble de règles très grand, se rapprochant de n. Dans les meilleurs cas, ces règles inutiles sont supprimées lors de la phase de pruning mais dans d'autres cas, l'ensemble de règles peut se retrouver extrêmement grand. Enfin, lors de la phase de pruning, REP supprime des règles et des conditions de la théorie générée. Cependant, comme le modèle contient beaucoup de règles et de conditions, la phase de pruning risque d'être lente et de s'arrêter à un maxima local plutôt qu'au maxima global.

Dans (Fürnkranz and Widmer 1994), Fürnkranz et Widmer présentent donc ce nouvel algorithme IREP qui se veut être une amélioration de REP. Dans cet algorithme, les méthodes de pre-pruning et de post-pruning sont utilisées conjointement. Au lieu de générer un modèle expliquant tous les exemples du dataset et de le restreindre par après, IREP propose de supprimer directement des conditions après la génération de chaque règle. Cette suppression est réalisée de manière *greedy* jusqu'à ce que n'importe quelle autre suppression de condition diminue la performance de la règle sur le pruning set. Cette performance de la règle est calculée par la fonction

$$v(Rule, PrunePos, PruneNeg) \equiv \frac{p + (N - n)}{P + N} \tag{3.1}$$

avec $PrunePos$ les instances positives du pruning set, $PruneNeg$ les instances négatives du pruning set, P le nombre total d'instances dans $PrunePos$, N le nombre total d'instances dans $PruneNeg$, p le nombre d'instances dans $PrunePos$ couvertes par $Rule$ et n le nombre d'instances dans $PruneNeg$ couvertes par $Rule$. Après le *pruning* de la règle, toutes les instances positives et négatives sont supprimées du growing set *et* du pruning set. Entre toutes les instances restantes,

```
procedure I-REP (Pos, Neg, SplitRatio)

Clauses = ∅
while Pos ≠ ∅
        SplitExamples(SplitRatio, Pos, PosGrow, PosPrune)
        SplitExamples(SplitRatio, Neg, NegGrow, NegPrune)
        Clause = ∅
        while NegGrow ≠ ∅
                Clause = Clause ∪ FindLiteral(Clause, PosGrow, NegGrow)
                PosGrow = Cover(Clause, PosGrow)
                NegGrow = Cover(Clause, NegGrow)
        Clause = PruneClause(Clause, PosPrune, NegPrune)
        if Accuracy(Clause) ≤ Accuracy(fail)
           return(Clauses)
        else
           Pos = Pos − Cover(Clause, Pos)
           Neg = Neg − Cover(Clause, Neg)
           Clauses = Clauses ∪ Clause
return(Clauses)
```

Figure 3.2: Algorithme IREP, tiré à partir de (Fürnkranz and Widmer 1994)

une séparation est faite pour obtenir de nouveaux growing set et pruning set. Enfin, si une règle réduite fait baisser la performance globale du modèle sur le pruning set, elle n'est pas ajoutée à l'ensemble de règles et IREP renvoie toutes les règles générées comme étant le modèle final.

Cette méthodologie permet de réduire ou supprimer certains problèmes inhérents à l'algorithme REP. En effet, la complexité de IREP ne sera que de $\Omega(\mathrm{nlog}^2 n)$ contre $\omega(\mathrm{n}^4)$ pour REP. Comme de nouveaux growing et pruning sets sont générés après l'addition de chaque règle, une mauvaise séparation n'affecte que l'apprentissage d'une seule règle. Cependant, une séparation malchanceuse pourrait générer une règle faisant baisser la performance générale du modèle et donc arrêter l'apprentissage prématurément. De plus, étant donné que le pre-pruning est utilisé pour IREP, tous les problèmes concernant la méthode *separate-and-conquer* évoqués pour REP ne peuvent pas survenir. Enfin, la méthode IREP est légèrement moins sensible aux minimas locaux que REP.

Néanmoins, une des grandes faiblesses de IREP reste son penchant vers l'underfitting. En effet, étant donné qu'une règle diminuant la performance totale du modèle est la condition d'arrêt de la phase d'apprentissage, si cette performance est estimée de manière erronée, la phase d'apprentissage s'arrête prématurément. Cela est principalement le cas lors d'une mauvaise séparation des données

ou lorsque trop peu d'instances sont présentes dans le pruning set. Cet arrêt précipité de l'apprentissage a pour effet de générer un nombre trop faible de règles et donc d'obtenir un modèle trop général. En faisant référence à la terminologie de (Schaffer 1993), IREP a un fort *Overfitting Avoidance Bias* ou un *Biais de prévention d'overfitting* en français.

3.1.2 RIPPERk

RIPPERk ou *Repeated Incremental Pruning to Produce Error Reduction* est un algorithme développé dans (William W Cohen 1995). Cette méthode se veut une amélioration de IREP présenté dans la section 3.1.1. En effet, dans son article Cohen a analysé les performances d'IREP et y a decelé des endroits améliorables. Il a donc modifié 3 éléments de IREP pour obtenir l'algorithme RIPPERk : la métrique pour évaluer la performance des règles, la condition d'arrêt de l'algorithme et une optimisation des règles a posteriori.

Métrique d'évaluation de la performance des règles

Dans son système d'évaluation des règles (voir 3.1), IREP préfère une règle couvrant 2000 exemples positifs et 1000 exemples négatifs plutôt qu'une règle couvrant 1000 exemples positifs et 1 exemple négatif. RIPPERk, quant à lui, va plutôt utiliser la fonction

$$v * (Rule, PrunePos, PruneNeg) \equiv \frac{p-n}{p+n} \tag{3.2}$$

pour évaluer la performance d'une règle *Rule*.

Condition d'arrêt

La condition d'arrêt d'IREP était déjà considérée comme une faiblesse par Fürnkranz et Widmer dans leur article le présentant (Fürnkranz and Widmer 1994). Elle a le défaut d'arrêter souvent l'algorithme trop tôt durant la phase d'apprentissage. C'est pour cette raison que Cohen a modifié cette condition d'arrêt. Dans IREP, si une règle fait baisser la performance du modèle sur le pruning set, elle n'est pas ajoutée à l'ensemble de règles et la phase d'apprentissage prend fin. Avec RIPPERk, la *longueur descriptive* de l'ensemble de règles et des instances doit être calculé. L'algorithme arrête d'ajouter des règles quand la longueur descriptive dépasse de d bits la plus petite longueur descriptive obtenue par avant. La valeur de d a été fixée à 64 lors des tests effectué par Cohen (William W Cohen 1995). Cette longueur descriptive est la même que celle utilisée dans

la dernière version de C4.5rules (Quinlan 1995). Cet algorithme évolué d'IREP avec la modification de la métrique d'évaluation des règles et la modification de la condition d'arrêt est appelé IREP* dans (William W Cohen 1995) et dans le reste de ce chapitre.

Optimisation de l'ensemble de règles

Après avoir généré un ensemble de règles avec IREP*, il est possible de l'améliorer en modifiant à nouveau chacune des règles induites. Cohen propose la méthode suivante pour optimiser cet ensemble de règles décisionnnelles. Chaque règle R_i est considérée dans l'ordre dans lequel elles ont été apprises. Pour chacune des règles, deux règles alternatives sont construites : le remplacement de R_i et la révision de R_i. Le remplacement $R_i^{'}$ est construit en partant d'une règle vide (i.e. sans aucune condition). Elle est ensuite réduite pour minimiser l'erreur de l'ensemble de règles $R1, ..., R_i^{'}, ..., R_k$. La révision de R_i est obtenue de façon analogue mais en démarrant de R_i et non pas d'une règle vide. Enfin, une décision est prise sur laquelle des trois règles inclure dans l'ensemble de règles : la règle originale, son remplacement ou sa révision. Cette décision est prise en utilisant la même heuristique que dans la sous-section 3.1.2, c'est-à-dire la longueur descriptive minimale de l'ensemble de règles.

Algorithme RIPPERk

Pour synthétiser l'algorithme RIPPERk, un ensemble de règles est tout d'abord généré grâce à l'algorithme IREP*. Ensuite, cet ensemble de règles est optimisé comme décrit dans la sous-section 3.1.2. Enfin, des dernières règles sont ajoutées en utilisant IREP* pour couvrir les dernières instances positives. Ces deux dernières étapes peuvent également être répétées plusieurs fois. Deux optimisations successives seraient appelées RIPPER2 et, en généralisant, RIPPERk répète ce processus d'optimisation de l'ensemble de règles k fois.

3.2 Algorithmes d'induction d'arbres de décision à partir de règles décisionnelles

De la même manière que plusieurs algorithmes génèrent des arbres de décision sur base de données stockées dans des datasets, il existe également certains algorithmes qui génèrent des arbres de décision en prenant comme input des règles décisionnelles. Les objectifs de ces méthodes sont d'ailleurs similaires avec les objectifs des algorithmes induisant des règles de décision.

Le but principal de ces procédés est de classer de manière optimale les instances du dataset. Comme pour les modèles induisant des règles de décision, le pourcentage d'instances correctement classifiées par celui-ci est un indicateur intéressant. Toutefois, une matrice de confusion peut être plus pertinente dans certains cas. Le deuxième objectif de ces méthodes concerne la simplicité de l'arbre. Il est important de ne pas générer un arbre contenant un trop grand nombre de noeuds pour éviter des problèmes d'*overfitting*, de lisibilité et d'interprétabilité. Certains méta-paramètres permettent de faire varier la complexité du modèle et donc de restreindre la grandeur de l'arbre. Il est notamment possible de sélectionner une profondeur limite de l'arbre, un nombre limite de noeuds ou encore un nombre minimum d'instances pour pouvoir subdiviser un noeud (en-dessous d'un certain nombre d'instances appartenant au noeud, il n'est plus subdivisé et est considéré comme une feuille).

Toutefois, tous ces méta-paramètres permettant de contrôler la complexité d'un arbre de décision ne sont pas toujours réglables par l'utilisateur. En effet, en utilisant des algorithmes ayant comme input des données, comme ID3 par exemple (Quinlan 1983), il est souvent possible de spécifier une valeur pour ces méta-paramètres. Cependant, les algorithmes se basant sur des règles décisionnelles pour générer un arbre de décision ne permettent pas de sélectionner des valeurs précises pour la profondeur limite de l'arbre, le nombre maximal de noeuds ou d'autres méta-paramètres inhérents aux arbres de décision.

3.2.1 AQDT

Imam et Michalski présentent l'algorithme AQDT dans leur article (Imam and Michalski 1993). Cet algorithme prend en input un ensemble de règles disjoint et produit un arbre de décision en output. L'utilisation de cet algorithme nécessite un prérequis important : tous les attributs doivent être catégoriels. Ce processus d'induction d'AQDT se déroule itérativement en répétant les deux étapes suivantes : la sélection d'un attribut et la génération de branches en fonction du nombre de valeurs possibles de l'attribut. Une représentation de l'algorithme est présenté à la figure 3.3

.

La sélection d'un attribut se déroule de la manière suivante. Chaque attribut est évalué en fonction de trois critères différents : la disjointeté, la dominance et l'étendue. L'attribut optimal est celui dont le degré de disjointeté est maximal. Si plusieurs attributs ont un degré de disjointeté équivalent, la dominance les départage. De même, si plusieurs attributs ont un degré de disjointeté

```
Procedure AQDT (Ruleset, Attributes, CurrentNode = None, Nodes = []):
while CurrentNode is not Leaf:
       Attribute = SelectBestAttribute<Disjointness/Dominance/Extent> (Ruleset, Attributes)
       for value in Attribute.values:
              Node = CreateNode(Attribute, value)
              Nodes.addNode(Node)
              Subruleset = getRules(Ruleset, Attribute, value)
              AQDT(Subruleset, Attributes, Node, Nodes)
return (Nodes)
```

Figure 3.3: Algorithme AQDT tiré à partir de Imam and Michalski 1993

et de dominance égaux, c'est leur degré d'étendue qui les départage. Ces trois critères se calculent sur base du sous-ensemble de règles de décision correspondant au noeud actuel. Si c'est l'attribut du premier noeud qui doit être sélectionné, alors tout l'ensemble de règles décisionnelles est pris en compte. Si c'est l'attribut d'un autre noeud qui doit être déterminé, alors ces critères sont calculés sur l'ensemble de règles correspondant à ce noeud.

La disjointeté de classe est définie dans (Imam and Michalski 1993) de la manière suivante : "Le degré de disjointeté $D(A, C_i)$ de l'attribut A pour l'ensemble de règles de classe C_i est la somme des degrés de disjointeté $D(A, C_i, C_j)$ entre l'ensemble de règles pour C_i et l'ensemble de règles pour C_j avec $j = 1, 2, ..., m, j \neq i$. Ce degré de disjointeté entre les ensembles pour C_i et C_j est défini par

$$D(A, Ci, Cj) = \begin{cases} 0, & \text{if } V_i \subseteq V_j \\ 1, & \text{if } V_i \supset V_j \\ 2, & \text{if } V_i \cap V_j \neq \emptyset \text{ or } V_i \text{ or } V_j \\ 3, & \text{if } V_i \cap V_j = \emptyset. \end{cases} \tag{3.3}$$

avec m, le nombre de classes différentes, V_i, l'ensemble des valeurs différentes de l'attribut A pour les règles appartenant à la classe C_i et V_j l'ensemble des valeurs différentes de l'attribut A pour les règles appartenant à la classe C_j. Le degré de disjointeté pour l'attribut A est défini comme la somme des degrés de disjointeté de classe pour chaque classe. On formalise cette relation par

$$D(A) = \sum_{i=1}^{m} D(A, C_i) \tag{3.4}$$

avec

$$D(A, C_i) = \sum_{i=1, i \neq j}^{m} D(A, C_i, C_j) \tag{3.5}$$

Au cas où la disjointeté ne suffirait pas à départager les différents attributs, la dominance est calculée pour les attributs ayant le degré de disjointeté maximal. Cette dominance est égale au nombre de règles dans lesquelles est présent l'attribut en question. AQDT préfère des attributs appartenant à un nombre important de règles. Dans certaines règles où les valeurs d'une condition sont multiples, il est nécessaire de compter plusieurs fois ces règles dans le calcul de la dominance. Par exemple, la règle

$$\text{IF } weather = [sunny, cloudy] \text{ AND } wind = [weak] \text{ THEN } Yes$$

doit être subdivisée dans les deux règles

- IF $weather = [sunny]$ AND $wind = [weak]$ THEN Yes
- IF $weather = [cloudy]$ AND $wind = [weak]$ THEN Yes

dans le calcul de la dominance.

Enfin, si plusieurs attributs ont un degré de dominance et de disjointeté équivalents, l'étendue de ces attributs est calculée. L'étendue d'un attribut est tout simplement égale au nombre de valeurs possibles que l'attribut peut prendre. Il est préférable pour un attribut d'avoir le moins de valeurs possibles. Cela a pour but de minimiser le nombre de branches sortant des noeuds pour limiter la taille de l'arbre de décision.

AQDT peut également prendre en compte le coût des attributs si cette option est activée par l'utilisateur. Dans ce cas-là, le coût des attributs est le critère prépondérant lors de la sélection de l'attribut optimal, avant même de calculer la disjointeté, la dominance et l'étendue. AQDT sélectionne donc premièrement l'attribut au coût le plus bas et uniquement si certains attributs ont le même coût, leur disjointeté respective est calculée. Inclure le coût des attributs comme critère de sélection de ceux-ci peut avoir une grande importance en pratique. En effet, dans certains domaines, pouvoir déterminer la valeur d'un attribut pour une instance peut coûter très cher. Une différence de coût entre les différents attributs peut notamment apparaître dans le secteur médical. Deux attributs possiblement utiles pour un médecin sont la température du patient et le volume globulaire moyen de son sang. Peut-être que le volume globulaire moyen est plus utile que la température pour

le médecin pour détecter certaines maladies. Toutefois, avoir la valeur de cet attribut nécessite une prise de sang qui implique une analyse en laboratoire et plusieurs jours de délai. La température du patient, d'autre part, est un attribut beaucoup moins *coûteux* étant donné que déterminer sa valeur ne nécessite qu'un thermomètre et quelques secondes d'attente. Il est donc parfois utile de commencer par évaluer les attributs dont le coût est minimal.

Après avoir sélectionné l'attribut optimal pour le noeud en question, plusieurs branches sont générées: pour chaque valeur possible de l'attribut, une branche est dérivée du noeud. Un attribut binaire générerait donc deux branches pour le noeud. L'ensemble de règles appartenant à chaque branche est ensuite déduit pour continuer à développer l'arbre de décision. Si toutes les règles appartenant à une des branches pointent toutes vers la même classe, une feuille est créée. Sinon, c'est un noeud qui est généré. Pour chaque nouveau noeud, il convient donc de répéter le processus de sélection de l'attribut optimal. Ce procédé s'arrête lorsque tous les noeuds ont été développés jusqu'à n'obtenir que des feuilles.

3.2.2 RBDT

RBDT est un algorithme de génération d'arbre de décision présenté dans l'article (Abdelhalim, Traore, and Sayed 2009). Tout comme AQDT, RBDT a besoin d'un ensemble disjoint de règles décisionnelles. Toutefois, RBDT exige également que cet ensemble de règles décisionnelles soit complet. Comme expliqué dans la section 2.4, un ensemble complet et disjoint implique que chaque instance appartient à une et une seule règle de décision. Ce prérequis pour l'utilisation de RBDT est assez contraignant car il n'est pas trivial de transformer un ensemble de règles non complet en un ensemble complet. Quand le nombre de règles et d'attributs est grand, rendre l'ensemble complet risque de faire augmenter encore plus le nombre de règles et potentiellement la taille de l'arbre de décision.

Le fonctionnement de l'algorithme RBDT est similaire à celui de AQDT. En effet, le processus de génération de l'arbre de décision se divise en les deux mêmes étapes qu'AQDT : la sélection d'un attribut et la génération des branches. Cependant, les critères de sélection de l'attribut ne sont pas les mêmes pour les deux algorithmes. Le premier critère de sélection de RBDT est appelé *attribute effectiveness*. La logique derrière ce critère est la suivante: si un attribut n'intervient pas dans les règles de décision d'une même classe, il n'est pas intéressant de le choisir comme l'attribut du

noeud. En effet, si un attribut n'intervenant dans aucune des règles d'une classe est choisi comme attribut d'un noeud, toutes ces règles sont présentes dans chaque branche induite du noeud. Dès lors, ce noeud n'a aucun impact sur les règles de cette classe.

Pour calculer ce critère, RBDT change légèrement la notation des règles des décisions en spécifiant la valeur DC en référence à *"Don't care"* aux attributs n'intervenant pas dans une règle de décision. RBDT nous présente ensuite une valeur de *Don't care* appelée $C_{ij}(DC)$ pour l'attribut a_j et l'ensemble de règles R_i correspondant à la classe C_i. Cette valeur est définie par l'équation

$$C_{ij}(DC) = \begin{cases} 0, & if DC \in V_{ij} \\ 1, & \text{otherwise.} \end{cases} \quad (3.6)$$

avec V_{ij}, l'ensemble des valeurs possibles pour l'attribut a_j dans l'ensemble de règles R_i. L'attribute effectiveness $AE(a_j)$ est ensuite calculé par

$$AE(a_j) = \frac{m - \sum_{i=1}^{m} C_{ij}(DC)}{m} \quad (3.7)$$

avec m le nombre de classes différentes dans l'ensemble de règles décisionnelles.

Comme pour l'algorithme AQDT présenté dans la section précédente, si plusieurs attributs ont un coefficient d'*attribute effectiveness* égal, d'autres critères seront calculés pour départager ces attributs. Dans l'ordre, ce seront l'*attribute autonomy* et la *minimal value distribution.*

L'*attribute autonomy* se base sur la disjointeté (appelée ADS dans l'article (Abdelhalim, Traore, and Sayed 2009)) qui a été décrite dans la section précédente (voir l'équation 3.3). Pour l'ensemble de règles R_{ij} où l'attribut a_j a la valeur v_{ij}, on note par $MaxADS_{ji}$ la valeur maximum que peut obtenir la disjointeté ADS et par ADS_List_{ji} la liste des scores de disjointeté pour chaque attribut a_k. Enfin, l'*attribute autonomy* est définie par l'équation

$$AA(a_j) = \frac{1}{\sum_{i=1}^{p_j} AA(a_j, i)} \quad (3.8)$$

```
Procedure RBDT (Ruleset, Attributes, CurrentNode = None, Nodes = []):
while CurrentNode is not Leaf:
      Attribute =
SelectBestAttribute<AttributeDisjointness/AttributeAutonomy/MinimalValueDistribution> (Ruleset,
Attributes)
      for value in Attribute.values:
            Node = CreateNode(Attribute, value)
            Nodes.addNode(Node)
            Subruleset = getRules(Ruleset, Attribute, value)
            RBDT|(Subruleset, Attributes, Node, Nodes)
return (Nodes)
```

Figure 3.4: Algorithme RBDT

avec

$$AA(a_j, i) = \begin{cases} 0, if \text{MaxADS}_{ji} = 0 \\ 1, if \Big(\Big(MaxADS_{ji} \neq 0 \Big) \wedge \Big((s = 2) \vee \Big(: MaxADS_{ji} = ADS\_List_{ji}[l] \Big) \Big) \Big) \\ 1 + \Big[(s-1) \times MaxADS_{ji} - \sum_{l=1, l \neq j}^{s} ADS\_List_{ji}[l] \Big], \text{otherwise.} \end{cases}$$

(3.9)

Enfin, la *minimum value distribution* correspond au nombre de valeurs différentes pour l'attribut en question dans les règles en question. Ce critère est le même que le critère de l'étendue dans l'algorithme AQDT, et de la même façon, les attributs avec le score minimum sont privilégiés. Cela a pour conséquence de diminuer le nombre de branches sortant du noeud.

En ce qui concerne le développement de l'arbre et la génération des branches, l'algorithme RBDT reprend le processus développé dans (Imam and Michalski 1993) pour AQDT. Pour chaque valeur différente de l'attribut, une branche est générée. Si toutes les règles appartenant à une branche appartiennent à la même classe, une feuille est déduite. Sinon, un noeud est ajouté au bout de la branche et le processus de sélection d'attribut continuera avec ce noeud. L'algorithme RBDT est représenté sur la figure 3.4.

3.3 Conclusion

Ce chapitre se conclut après avoir présenté deux types d'algorithmes : ceux induisant des règles de décision et ceux induisant un arbre de décision sur base de règles décisionnelles. IREP et sa version améliorée RIPPERk permettent de produire un ensemble de règles décisionnelles en prenant des données comme input. AQDT et RBDT, quant à eux, produisent un arbre de décision en prenant un ensemble de règles comme input. Un ensemble disjoint est nécessaire pour l'utilisation d'AQDT tandis que l'ensemble de règles doit également être complet pour pouvoir utiliser RBDT.

Chapitre 4

Enquête qualitative

Les différents algorithmes de la littérature étant présentés, il est désormais temps de passer à une enquête qualitative durant laquelle plusieurs experts en data science et machine learning sont questionnés. Ces interviews ont un objectif principal : déterminer si la méthodologie développée et présentée dans la section 5 est cohérente, utile et pertinente pour une utilisation dans un contexte industriel. Pour vérifier si cela est bien le cas, une série de questions sont posées aux experts concernant tant leur expérience passée en machine learning que leur avis sur la méthodologie. Le questionnaire est présenté dans la section suivante 4.1. En ce qui concerne les différentes personnes interviewées, elles ont comme point commun d'avoir une expertise très poussée en machine learning. De plus, elles ont pu travailler sur des projets dans des domaines extrêmement variés. L'actuariat, la médecine, la sécurité informatique, l'industrie aéronautique et ferroviaire sont, entre autres, des domaines dans lesquels au moins un des experts a pu expérimenter un projet de machine learning.

4.1 Présentation du questionnaire

Pour pouvoir atteindre l'objectif de cette enquête qualitative, un questionnaire a été pensé et réfléchi pour être aussi ouvert que possible. Ainsi, les interviews se rapprochent plus d'une discussion ouverte que d'une suite de questions fermées dont les réponses se limiteraient à quelques choix présélectionnés. Ce questionnaire est divisé en trois parties différentes : les contraintes techniques, les contraintes opérationnelles (ou métier) et la présentation d'un exemple.

Les questions sur la partie contraintes techniques sont présentes à l'annexe A.1 tandis que les questions sur la partie des contraintes opérationnelles se retrouvent à l'annexe A.2. Enfin la troisième

section de l'interview présente la mise en situation suivante. Lors d'une demande de prêt, un banquier désire classifier les demandeurs en fonction de leur risque de non-remboursement du prêt. Pour cela, il a un dataset composé de clients précédents avec plusieurs caractéristiques comme l'âge, le salaire, le montant du compte en banque, etc. Pour chacun des clients, le banquier sait également s'il a remboursé ou pas son prêt. Avec ce dataset, il espère un arbre de décision lui permettant de prédire le plus précisément possible si de futurs clients rembourseront ou pas le prêt. Deux méthodes pour réaliser l'arbre de décision sont présentées aux experts interviewés. La première méthode est assez *classique* : un arbre de décision est construit directement avec un algorithme de machine learning se basant sur le dataset (i.e. ID3). En annexe B.1, se trouve un possible arbre de décision construit avec cette première méthode. Dans la mise en situation, le banquier n'est pas à 100% satisfait car il aimerait inclure la contrainte suivante dans l'arbre de décision :

$$\text{IF } Salary < 1200 \text{ THEN } Repayment = No$$

Cette contrainte peut être mise en place pour différentes raisons. L'expérience du banquier peut lui indiquer que la plupart des clients dont le montant du salaire est inférieur à 1200€ ne remboursent pas leur prêt. Cela peut être également une assurance pour le banquier de ne pas accorder de prêt aux personnes ayant moins de 1200€ de revenus mensuels. Enfin, cette contrainte peut être une exigence du patron du banquier. Dans tous les cas, inclure cette contrainte dans l'arbre de décision peut se révéler compliqué. De plus, plus ce nombre de contraintes augmente et plus il est difficile de les inclure dans l'arbre de décision. Une deuxième méthode est ensuite présentée pour répondre à la problématique du banquier. A partir du dataset, un ensemble de règles de décision est généré (voir annexe B.2). Cet ensemble de règles peut être modifié pour ajouter la contrainte du banquier sous la forme d'une règle de décision. Les modifications de l'ensemble sont les suivantes :

- Ajout de la règle : IF $Salary < 1200$ THEN $Repayment = No$
- Suppression de la règle : IF $Salary < 1200 \wedge Age < 25$ THEN $Repayment = No$

L'ensemble modifié peut être consulté à l'annexe B.3. Après cette modification, un arbre de décision est généré par l'algorithme *AQDT*. Cet arbre est présent à l'annexe B.4.

Après la présentation de cet exemple, les experts interviewés se sont vus questionnés par rapport à celui-ci. La liste des questions est disponible à l'annexe B.5.

4.2 Idées ressorties des interviews

Beaucoup d'idées et de conclusions peuvent être tirées des interviews réalisées. Elles sont donc séparées en plusieurs catégories : la présentation des experts, les contraintes et l'avis sur la méthodologie.

4.2.1 La présentation des experts

Les différents experts viennent de domaines très différents. Évidemment, cette hétérogénéité des domaines d'expertise était désirée pour obtenir des avis couvrant le plus d'industries différentes. La plupart des experts utilisent beaucoup d'algorithmes de machine learning différents en fonction de la problématique rencontrée. Les réseaux de neurones et les forêts d'arbre de décision sont les deux algorithmes les plus souvent mentionnés par les experts. Toutefois, les arbres de décision sont assez souvent utilisés dans l'industrie pour certains sous-systèmes. Ces arbres de décision sont surtout utiles lorsque l'interprétabilité et l'explicabilité sont les objectifs principaux de l'algorithme de machine learning. En ce qui concerne les différents types de données, les réponses sont également très variées. Certains experts ont affaire à des séries temporelles (venant de capteurs, d'électroencéphalogrammes, de réseaux DNS,...), des images, des textes, etc.

4.2.2 Les contraintes

Les contraintes dont les experts ont fait part dans leurs réponses sont principalement de deux types : les contraintes influençant le choix de l'algorithme de machine learning et les contraintes sur le modèle en tant que tel. Ce premier type de contraintes concerne par exemple l'objectif final du modèle. Si ce but est principalement la transparence, l'interprétabilité ou l'explicabilité du modèle, certains algorithmes sont privilégiés tandis que si la performance est le seul objectif, d'autres algorithmes sont sélectionnés. Une autre contrainte également mentionnée par un expert est le fait de pouvoir *formuler la problématique de façon informatique.*

Pour le deuxième type de contraintes, les experts ont principalement fait part de la *feature selection.* En effet, dans beaucoup de projets de machine learning, il est nécessaire de procéder à une sélection d'un sous-ensemble des features disponibles pour alimenter le modèle. Cette sélection est d'ailleurs particulièrement utile lorsque l'exigence principale concernant le modèle reste l'explicabilité de celui-ci. Un autre argument prônant une sélection des features avant l'entraînement du modèle est le *federated learning.* Cette méthode a vu le jour en réponse aux nouvelles législations concernant la

protection des données personnelles. Il est dans certains cas nécessaire de ne pas pouvoir retrouver à quel individu appartient une combinaison de caractéristiques données. Pour cette raison, toutes les features critiques ne peuvent pas apparaître en même temps dans le même modèle. Par exemple, certaines features prises conjointement telles que l'âge, le sexe, la ville d'origine, la date de naissance et le niveau scolaire atteint risquent de ne plus pouvoir assurer l'anonymisation des données à caractère personnel. C'est dans ce cas que le federated learning est utilisé : plusieurs modèles sont construits en utilisant à chaque fois un sous-ensemble des features disponibles.

Il est peu souvent arrivé aux experts questionnés de répondre à des contraintes opérationnelles. Aucun n'a mentionné avoir eu à faire à une problématique où certaines instances seraient plus importantes à classifier que d'autres. Toutefois, le coût monétaire nécessaire à l'obtention de certaines features est d'une importance cruciale dans les domaines industriels. Selon l'expérience de plusieurs experts, cette contrainte budgétaire sur les features disponibles est souvent présente et contraint le modèle de machine learning. Une autre contrainte opérationnelle mentionnée par les experts est le niveau d'exhaustivité des données. Certains datasets peuvent ne pas être assez exhaustifs, ne pas représenter toutes les situations possibles dans l'environnement analysé et cela pose problème lors de la phase de test du modèle.

Enfin, quelques contraintes techniques ont été citées par certains experts comme primordiales lors de la création de modèles de machine learning. En effet, le temps de calcul doit, dans des domaines spécifiques, être parfois très limité et un modèle moins performant mais plus rapide sera préféré à un modèle plus lent. C'est par exemple le cas pour la reconnaissance automatique de panneaux de signalisation pour les trains sur les voix ferrées. Un autre exemple de contrainte technique pour un modèle est sa *scalabilité*, ou sa capacité à performer peu importe la quantité de données. Si un modèle a de bons résultats avec peu d'instances, il doit également en avoir avec un nombre d'instances beaucoup plus élevé.

4.2.3 Avis sur la méthodologie

Après avoir présenté la méthodologie à travers un exemple (voir section 4.1), un avis concernant celle-ci a été demandé aux experts. Pour déterminer quelles sont les idées intéressantes et quelles sont les points à améliorer, plusieurs aspects de la méthodologie ont été séparés.

Le premier critère évalué est l'utilité d'un expert métier le long d'un processus de machine learning. Pour ce critère en particulier, l'avis des experts est unanime : il est important d'avoir l'expertise et l'expérience d'un expert du domaine lors de la création, l'entraînement et la phase de test d'un modèle de machine learning. Un exemple de ce que peut apporter un expert métier apparaît clairement dans des domaines techniques comme les traces réseaux analysées pour y détecter des comportements malicieux. Dans ce genre de projet, l'apport que peut avoir un expert ayant une connaissance aiguë des différents trafics DNS et des attaques possibles sur celui-ci est évident.

La pertinence et l'utilité de passer par des règles de décision est un second critère évalué. Cet aspect de l'approche est assez difficile à évaluer car aucun expert n'avait déjà expérimenté cette manière de générer un arbre de décision. Aucune des personnes interviewées n'avait entendu parler de ces deux algorithmes AQDT et RBDT qui permettent de générer des arbres de décision en partant de règles décisionnelles ni de RIPPERK. Ils ont donc pour la plupart admis que cette approche pouvait être intéressante mais également ne pas savoir si cette méthode est réellement efficace.

L'avis des experts a été demandé en ce qui concerne le biais possible inhérent de la méthodologie. En ce qui concerne cet aspect de l'approche, tous les experts s'accordent à dire que le changement du modèle après l'entraînement de celui-ci engendre du biais. L'expert métier va incorporer son propre avis, se basant sur son expertise et des heuristiques propres à chaque être humain. Cette action engendre donc toujours du biais dans l'arbre de décision final généré. Ce qui compte vraiment, selon les experts questionnés, c'est si l'inclusion de ce biais a une contrepartie assez forte pour le contrebalancer. Une des contreparties pourrait être une interprétabilité et une explicabilité plus grande de l'arbre de décision ou encore une amélioration de la performance lors des phases de test (dans le cas de données manquantes ou biaisées elles-mêmes).

Enfin, il a été demandé aux experts de critiquer la méthodologie en tant que telle en terme d'utilité, de pertinence et de cohérence. Pour cette question finale, l'avis des experts est assez partagé. La majorité des experts est assez enthousiaste et salue l'utilité de la méthodologie dans quelques cas précis. Pour eux, le principal avantage de cette approche reste la facilité d'interprétabilité du modèle. Ainsi, l'utilisateur de cette méthodologie pourrait facilement incorporer son expertise pour rendre l'arbre de décision final assez simple et interprétable. Toutefois, ils pointent également le fait qu'un changement des règles *humain* après l'entraînement du modèle apporte obligatoirement

du biais et que la performance du modèle doit toujours être prise en considération. Donc, pour ces experts, une légère baisse de performance, en contrepartie d'une simplification de l'arbre de décision, peut être utile et pertinente. Un des experts a également soulevé l'idée de la pertinence de cette approche dans le cas où les données sont peu nombreuses ou souffrent d'une forme de biais. Dans ce cas, l'expert du domaine peut *rectifier* le tir et guider le modèle. Cela pourrait éventuellement améliorer la performance du modèle lors de la phase de test sur des données inconnues.

Deux experts se sont montrés plus critiques et ne voient pas d'utilité et d'intérêt pour l'approche proposée. Ils voient cette intervention humaine après l'entraînement du modèle comme assez dangereuse. Selon eux, le risque existe de trop modifier l'arbre de décision et de complètement le dénaturer, d'oublier certaines branches importantes ou même de trop le complexifier.

Finalement, quelques points importants ont été soulevés par les personnes interviewées pour améliorer certains aspects. Tout d'abord, la performance de l'arbre de décision devrait toujours être indiquée (sous forme de pourcentage d'instances correctement classifiées) pour guider l'utilisateur dans sa modification des règles. De plus, les règles *inutiles* car déjà incluses dans d'autres règles devraient être automatiquement supprimées par le logiciel. La question de la transposabilité de la méthodologie pour d'autres algorithmes de machine learning que des arbres de décision est revenue également plusieurs fois. Enfin, le fait que l'approche présentée ne fonctionne pour l'instant qu'avec des variables catégorielles et uniquement pour des problématiques de classification est une limite importante pour quelques experts. Selon eux, cela ne représente pas les réels besoins business où des données et des variables de toutes sortes sont généralement nécessaires.

4.3 Conclusion

En résumé, cette enquête qualitative permet de déceler quels sont les points forts et les points faibles de l'approche présentée au chapitre suivant. Les résultats sont mitigés mais encourageants car plusieurs experts pointent l'utilité et la pertinence de la méthode. Toutefois, certains aspects comme l'introduction d'un fort biais par l'expert du domaine sont à surveiller lors de l'utilisation de la méthode dans un contexte industriel.

Chapitre 5

Présentation de la méthodologie

5.1 Introduction à la problématique

L'intelligence artificielle et le machine learning sont désormais des parties intégrantes de la société actuelle. Pourtant, plusieurs problématiques récurrentes apparaissent en utilisant des algorithmes de machine learning. Un des défis importants des modèles reste de traiter avec des datasets de mauvaise qualité. Cette qualité amoindrie du dataset peut résulter de différentes causes.

Premièrement, certaines instances du dataset peuvent contenir des valeurs manquantes ou incorrectes. Régler cette problématique est l'objectif de la partie du machine learning que l'on appelle *data preparation*. Pour ce qui est des valeurs manquantes, il est possible de ne pas prendre en compte les instances dont les valeurs de certains attributs sont inconnues. Il est également envisageable de remplacer les valeurs manquantes par la valeur majoritaire pour les autres instances dans le dataset. Toutefois, il reste difficile de gérer les données erronées dans un dataset.

Deuxièmement, les modèles peuvent être soumis à du biais. Dans (Mehrabi et al. 2021), un algorithme *juste* est défini comme un algorithme où il y a une "absence de tout préjudice ou favoritisme en faveur d'un individuel ou d'un groupe sur base de leurs caractéristiques inhérentes ou acquises". Dans ce même article, deux éléments sont identifiés comme sources de cette *injustice* : les biais provenant du dataset et les biais provenant de l'algorithme. Ces biais peuvent prendre de nombreuses formes. Les données peuvent être collectées sur un échantillon trop étroit de la population et ne pas représenter la *réalité* du domaine. Elles peuvent également contenir des données n'étant plus d'actualité. Il est également possible que la réalité soit complètement différente d'un endroit

géographique à un autre et que les données mélangeant différentes zones ne donnent pas un résultat représentatif. Cette problématique est assez difficile à résoudre car il est nécessaire pour cela de trouver le biais.

Enfin, un dataset peut ne pas contenir assez de données. Il est important de comprendre que les données ont un coût très variable en fonction du domaine de recherche. Réaliser une enquête médicale sur des milliers de patients coûte énormément d'argent en comparaison d'une enquête sur le prix des produits dans les supermarchés, par exemple. Il est donc parfois utopique d'espérer avoir à sa disposition des milliers d'instance sur lesquelles entraîner notre modèle.

Tous ces challenges auxquels fait face le machine learning peuvent parfois nuire aux résultats obtenus par les algorithmes. La méthodologie présentée dans cette section a pour humble objectif d'améliorer les résultats qu'un modèle de machine learning peut obtenir en ajoutant un expert dans le processus d'induction des règles de décision. Pour pouvoir utiliser tous les savoirs d'un ou plusieurs experts, la première étape de la méthodologie est l'induction d'un ensemble de règles grâce à un algorithme de machine learning dans la section 5.3. Dans la section, 5.4, il est présenté comment les experts peuvent améliorer la liste de règles induites. Enfin, la section 5.5 présente la méthode d'induction de l'arbre de décision final à partir des règles modifiées par les experts.

5.2 Conditions pour utiliser la méthodologie

L'approche développée nécessite quelques prérequis. Tout d'abord, seuls les attributs catégoriels sont admis dans le dataset de départ. Dans le cas où certains attributs ne seraient pas catégoriels mais continus, il convient de les discrétiser. Par exemple, si l'âge est un des attributs présents dans le dataset, il est possible de créer une catégorie par dizaine d'âge : [0-10[, [10-20[, etc. De plus, le dataset doit être sous la forme d'un document *.csv* avec la première ligne contenant le nom de tous les attributs (incluant le nom de l'attribut cible). Toutes les autres lignes contiennent les valeurs des attributs pour toutes les instances séparées les unes des autres par des virgules.

5.3 Induction d'un ensemble de règles

Pour l'induction d'un ensemble de règles, cette méthodologie utilise l'algorithme RIPPERk présenté dans la section 3.1.2. Les différents paramètres à introduire par l'utilisateur sont : le chemin vers le

document *.csv* contenant le dataset et le nom de la colonne cible dans le fichier *.csv*. L'algorithme RIPPER va être lancé autant de fois qu'il y a de classes présentes dans le dataset. Pour chacune de ces classes, un ensemble de règles est induit. L'ensemble de règles final est composé de l'ensemble des règles décisionnelles induites pour chacune des différentes classes. Il est intéressant de noter que cet ensemble de règles n'est ni disjoint ni complet. Cela signifie donc que plusieurs problèmes sont possibles, comme introduits dans la section 2.3. Tout d'abord, certaines instances peuvent appartenir à différentes règles dont les classes sont différentes. Il faut donc déterminer quelle règle prime sur les autres. Ensuite, certaines instances peuvent n'appartenir à aucune règle. Il est donc nécessaire de déterminer quelle classe prédire pour l'instance dans ces cas spécifiques.

5.4 Modification des règles

Après avoir induit un ensemble de règles avec l'algorithme RIPPERk, un expert du domaine peut provoquer des modifications sur ces règles. Plusieurs changements peuvent être réalisés : la modification d'une règle, l'addition d'une règle et la suppression d'une règle.

5.4.1 Modification d'une règle

L'expert peut modifier une règle de différentes manières. Tout d'abord, il peut enlever une des conditions de la règle pour la rendre plus générale (voir tableau 5.1).

Règle précédente	IF $outlook = sunny$ AND $wind = normal$ THEN Yes
Règle modifiée	IF $outlook = sunny$ THEN Yes

Table 5.1: Suppression d'une condition

La modification d'une règle peut également être réalisée en ajoutant une condition à celle-ci (voir tableau 5.2). Une troisième modification réalisable est la modification de la classe prédite

Règle précédente	IF $outlook = sunny$ AND $wind = normal$ THEN Yes
Règle modifiée	IF $outlook = sunny$ AND $wind = normal$ AND $humidity = normal$ THEN Yes

Table 5.2: Ajout d'une condition

par la règle (voir tableau 5.3). Enfin, l'expert du domaine peut également changer une règle en

Règle précédente	IF $outlook = sunny$ AND $wind = normal$ THEN Yes
Règle modifiée	IF $outlook = sunny$ AND $wind = normal$ THEN No

Table 5.3: Changement de la classe prédite par la règle

réalisant plusieurs des trois modifications possibles en même temps. Il peut par exemple ajouter une condition, en enlever une autre et changer la classe prédite par la règle.

5.4.2 Suppression d'une règle et ajout d'une règle

Dans certaines situations, un expert peut vouloir supprimer des règles de l'ensemble pour simplifier l'arbre de décision futur. Dans d'autres cas, l'ajout d'une règle pour exprimer une contrainte opérationnelle peut être utile. Pour ce faire, l'expert choisit pour chaque attribut la valeur qu'il désire ainsi que la classe à prédire. Pour les attributs, l'utilisateur peut également choisir la valeur DC pour indiquer que la valeur de cet attribut n'influence pas la prédiction de la règle.

5.5 Induction d'un arbre de décision

Pour induire un arbre de décision sur base d'un ensemble de règles décisionnelles, deux algorithmes sont présentés dans la littérature : AQDT et RBDT (voir section 3.2). AQDT présente un avantage conséquent par rapport à RBDT : l'ensemble de règles décisionnelles à fournir à l'algorithme ne doit pas forcément être complet. Toutefois, il est nécessaire pour utiliser l'algorithme AQDT de lui donner un ensemble de règles disjoint. Dès lors, il est indispensable de réaliser une étape intermédiaire avant d'utiliser AQDT : rendre disjoint l'ensemble de règles modifié par les experts.

5.5.1 Rendre l'ensemble de règles décisionnelles disjoint

Rendre un ensemble de règles décisionnelles disjoint est une manipulation complexe et peu discutée dans la littérature. Quand deux règles ne sont pas disjointes, il est nécessaire de modifier une des deux règles. Cette modification doit se faire obligatoirement en ajoutant une condition à l'une des deux règles de manière à ce qu'une instance ne puisse plus appartenir aux deux règles simultanément. Ci-dessous se trouvent deux règles fictives pour illustrer cette problématique :

- $A = 0 \Rightarrow C1$
- $B = 0 \Rightarrow C2$

Les valeurs possibles pour les deux attributs A et B sont $0, 1$. Ces deux règles ne sont pas disjointes car une instance telle que ses attributs A et B sont tous deux égaux à 0 appartient aux deux règles. Il existe donc deux façons possibles de modifier les deux règles pour qu'elles deviennent disjointes. La première façon est d'ajouter une condition $B = 1$ dans la première règle. Elle deviendrait donc

$$\text{IF } A = 0 \wedge B = 1 \text{ THEN } C1$$

La deuxième façon de modifier les règles est d'ajouter la condition $A = 1$ dans la deuxième règle. Elle deviendrait donc

$$\text{IF } A = 1 \wedge B = 0 \text{ THEN } C2$$

Cette méthodologie utilise la méthode suivante pour transformer l'ensemble de règles :

- 1. Trouver deux règles non disjointes dans l'ensemble de règles décisionnelles.
- 2. Identifier les différentes modifications possibles pour que les deux règles deviennent disjointes.
- 3. Pour chaque modification, évaluer le score du dataset résultant de la modification. L'évaluation de ce score est un parti pris de la méthodologie. Il est égal au pourcentage d'instances correctement classifiées par l'ensemble de règles sur l'ensemble du dataset. Cependant, il est nécessaire de déterminer quelle classe est prédite pour les instances appartenant à plusieurs règles et celles n'appartenant à aucune règle. Dans les deux cas, cette méthodologie utilise les conseils de Quinlan dans (Quinlan 1987). Quinlan préconise de prédire la classe majoritaire du dataset pour les instances n'appartenant à aucune des règles de décision. Pour les instances appartenant à plusieurs règles, Quinlan donne un facteur de certitude à chacune des règles. La règle dont le facteur de certitude prime sur les autres et l'instance devrait être classifiée comme le dit la règle en question. Ce facteur de certitude est calculé comme suit :

  $$\frac{np - 0,5}{np + nq}$$

  avec np le nombre d'instances correctement classifiées par la règle et nq le nombre d'instances classifiées de façon erronée.

- 4. Modifier l'ensemble de règles en fonction des scores obtenus au point 3. Si plusieurs modifications obtiennent un score équivalent, privilégier la modification induisant l'ensemble de règles contenant le moins de règles.
- 5. Recommencer les étapes 1 à 4 tant qu'il y a au moins deux règles non disjointes dans l'ensemble de règles décisionnelles.

5.5.2 Induire un arbre de décision

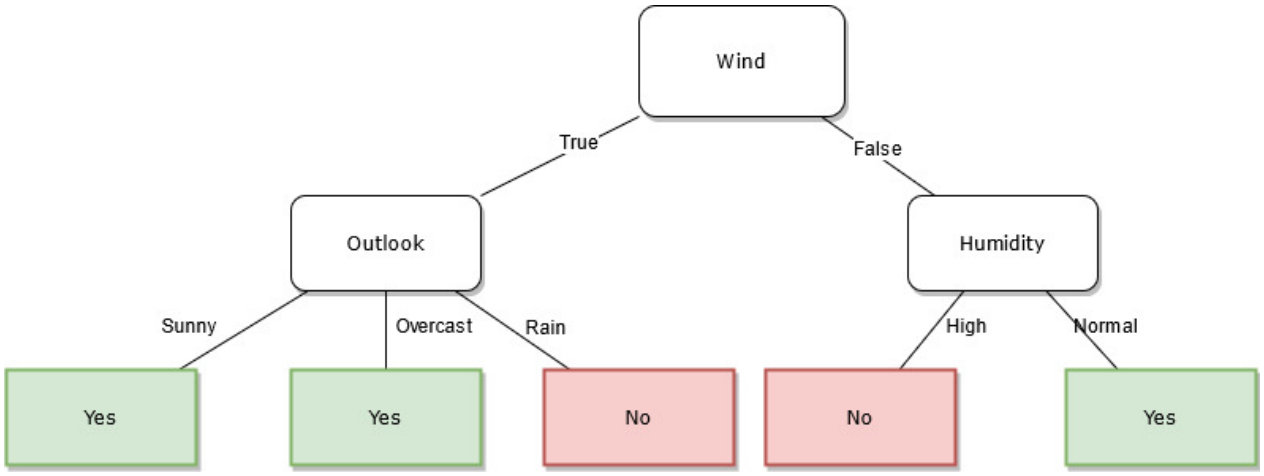
Lorsque l'ensemble des règles décisionnelles est totalement disjoint et qu'aucune instance ne peut appartenir à deux règles simultanément, l'algorithme AQDT (voir section 3.2.1) peut être utilisé pour induire un arbre de décision.

L'algorithme AQDT nous permet d'obtenir un arbre de décision représentant fidèlement l'ensemble des règles de décision. Toutefois, AQDT est légèrement modifié pour un souci de lisibilité et de réduction de taille de l'arbre de décision. Cette modification est la suivante : dans le cas où plusieurs branches découlant d'un même noeud génèrent des feuilles de même classe, elles seront rassemblées en une seule branche générant une seule feuille (voir figure 5.1b). Ce changement permet dans certains cas de réduire considérablement le nombre de feuilles et améliore la lisibilité de l'arbre de décision. Cet arbre généré par AQDT est un arbre non-binaire et plus de deux branches peuvent découler d'un même noeud.

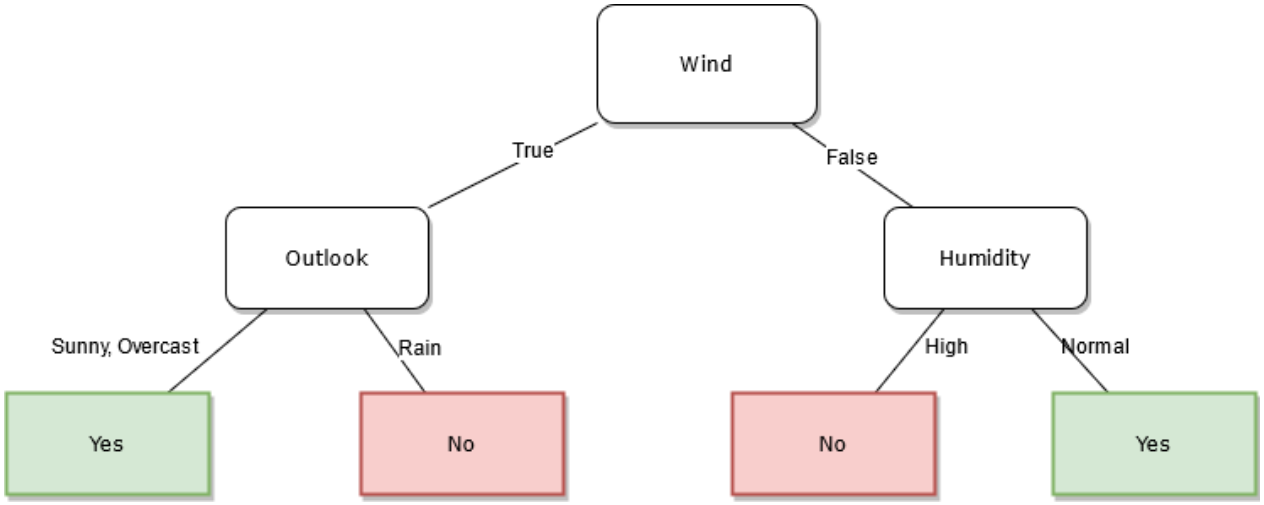
5.5.3 Visualisation de l'arbre de décision

Pour pouvoir visualiser l'arbre de décision, cette méthodologie utilise le package *graphviz* de python. Toutefois, ce package ne nous permet que de visualiser des arbres binaires. Il est donc nécessaire de transformer légèrement la structure de notre arbre de décision pour qu'il n'admettent qu'un maximum de branches à chaque noeud.

Sur la figure 5.2a est représenté un arbre de décision non-binaire car il y a trois branches attachées au premier noeud. Pour rendre cet arbre binaire, il est nécessaire d'ajouter un noeud supplémentaire, représenté en orange sur la figure 5.2b. De manière générale, pour chaque branche dépassant les



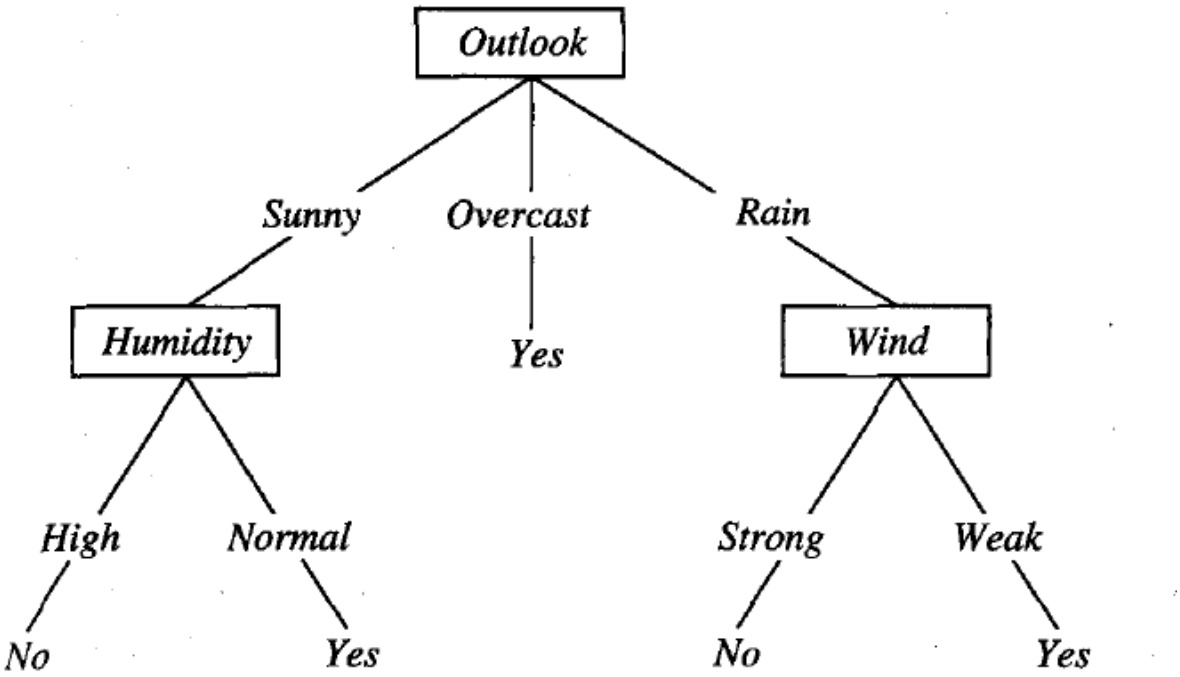
(a) Arbre de décision sans la modification d'AQDT



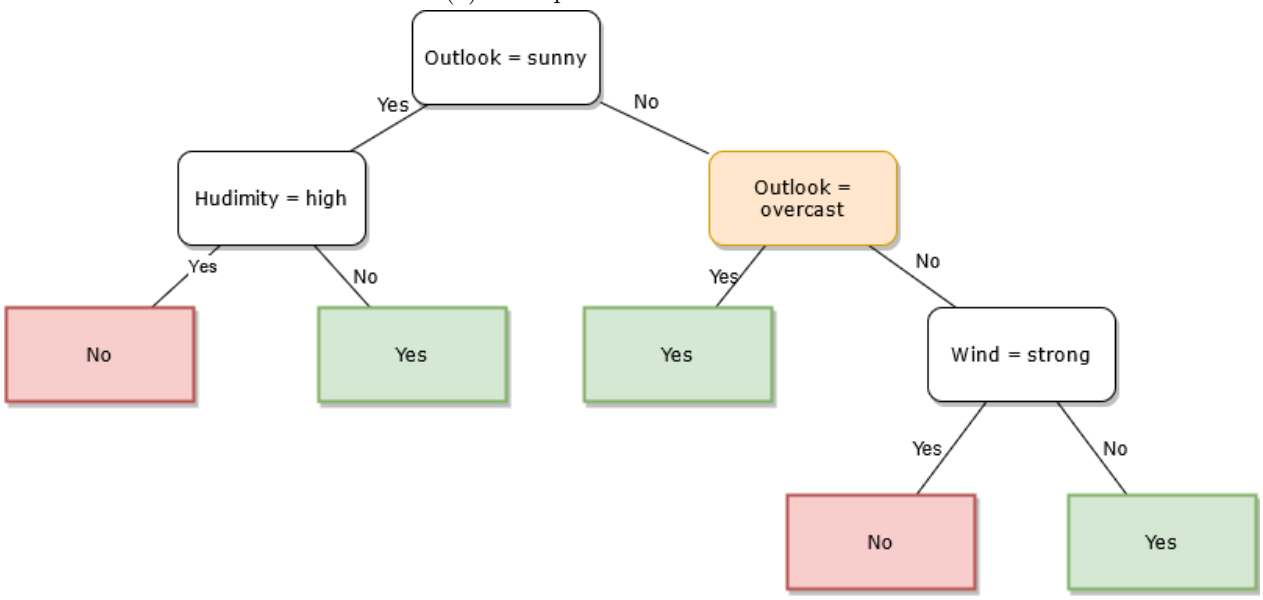
(b) Arbre de décision avec la modification d'AQDT

Figure 5.1: Arbres de décision avec et sans la modification d'AQDT

deux par noeud autorisées, il est nécessaire d'ajouter un noeud intermédiaire. Les arbres binaires seront donc toujours plus profonds que les arbres non-binaires correspondants.



(a) Exemple d'arbre non-binaire



(b) Exemple d'arbre binaire

Figure 5.2: Exemples d'arbres binaires et non-binaires

5.6 Conclusion

Toutes les étapes de la méthode sont présentées dans ce chapitre et la présentation de l'interface utilisateur peut suivre dans le chapitre suivant. Pour résumer la méthodologie, un ensemble de

règles est tout d'abord généré avec RIPPERk. Ensuite, l'utilisateur peut modifier cet ensemble de règles avec une ou plusieurs des 5 opérations possibles : l'ajout d'une règle, la suppression d'une règle, l'ajout d'une condition, la suppression d'une condition et le changement de règle pour une règle. Quand les modifications sont terminées, l'ensemble de règles est automatiquement transformé en un ensemble de règles disjoint utilisable par AQDT. Enfin, l'arbre de décision final est généré par l'algorithme AQDT.

Chapitre 6

Interface utilisateur

Après avoir introduit toute la méthodologie dans le chapitre précédent, ce chapitre voit la théorie mise en oeuvre, façonnée sous la forme d'un logiciel à destination des experts métiers. Ce logiciel est divisé en trois interfaces suivant le parcours du processus de machine learning : l'introduction des données, les modifications de l'utilisateur et la visualisation de l'arbre de décision.

6.1 Introduction des données

Avant que tout modèle puisse voir le jour, chaque algorithme de machine learning a besoin des données pour pouvoir s'entraîner. L'utilisateur du logiciel peut introduire ces données à travers un fichier *csv* à uploader. Le champ *target column* doit être spécifié par l'utilisateur pour désigner la colonne cible dans le dataset (voir figure 6.1).

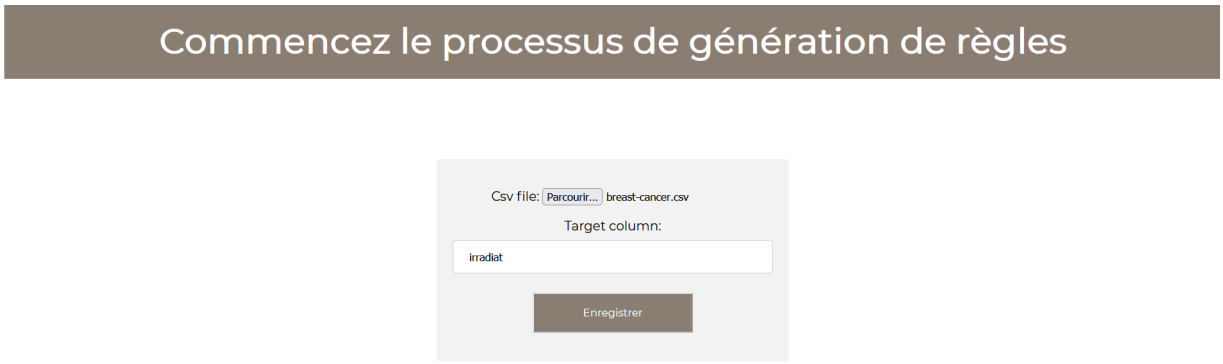


Figure 6.1: Interface utilisateur : introduction des données

6.2 Modifications des règles décisionnelles

Les données étant introduites dans le logiciel, RIPPERk génère un ensemble de règles décisionnelles présentées à l'utilisateur dans la deuxième interface. Cette page est le point de départ pour toutes les modifications que l'utilisateur voudrait faire. Il peut ajouter, supprimer ou modifier une ou plusieurs règles (voir figure 6.2).

Figure 6.2: Interface utilisateur : règles décisionnelles

L'ajout d'une règle se fait à partir d'un tableau où l'utilisateur choisit les différentes valeurs pour chacun des attributs. Il peut choisir uniquement une valeur par attribut, en sachant que la valeur *DC* représente *Don't care* ce qui signifie que la valeur de l'attribut n'influe pas dans la prédiction de la règle de décision. L'utilisateur choisit également parmi toutes les classes du dataset laquelle sera prédite par la règle créée (voir figure 6.3).

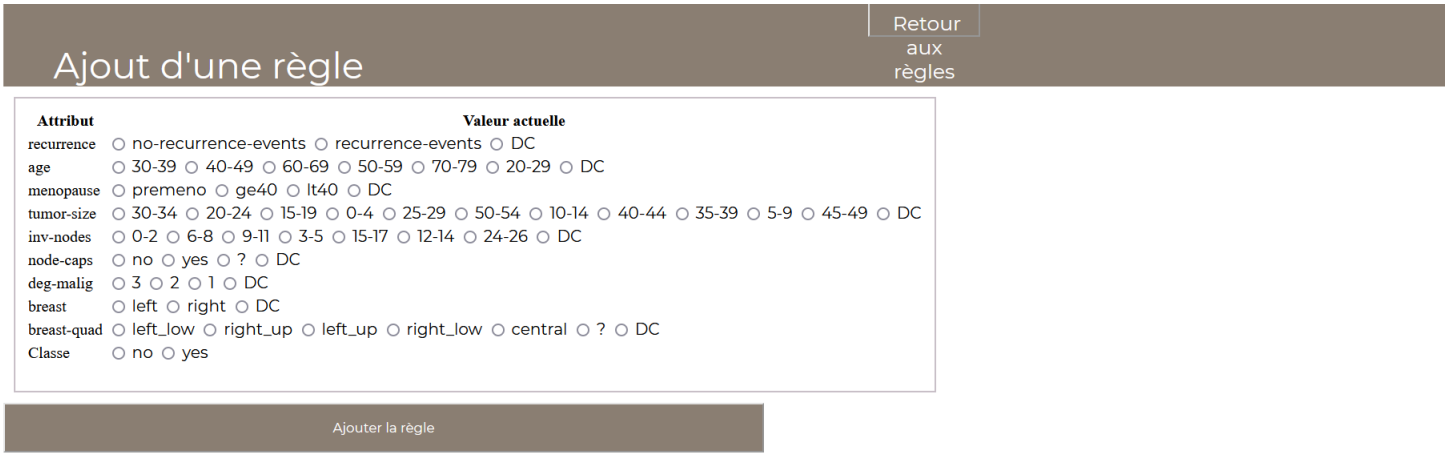


Figure 6.3: Interface utilisateur : ajout d'une règle

La suppression d'une règle doit être confirmée par l'utilisateur pour éviter les erreurs de manipulation qui engendreraient des suppressions de règles malencontreuses (voir figure 6.4).

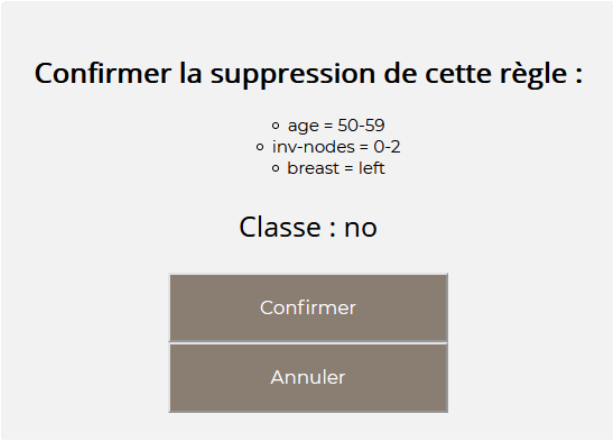


Figure 6.4: Interface utilisateur : suppression d'une règle

Enfin, les règles de décision peuvent être modifiées par l'expert du domaine. Pour cela, l'utilisateur

remplit un formulaire similaire à celui de l'ajout de règle. L'interface affiche les valeurs des attributs de la règle initiale à gauche du tableau comme indication pour l'utilisateur (voir figure 6.5).

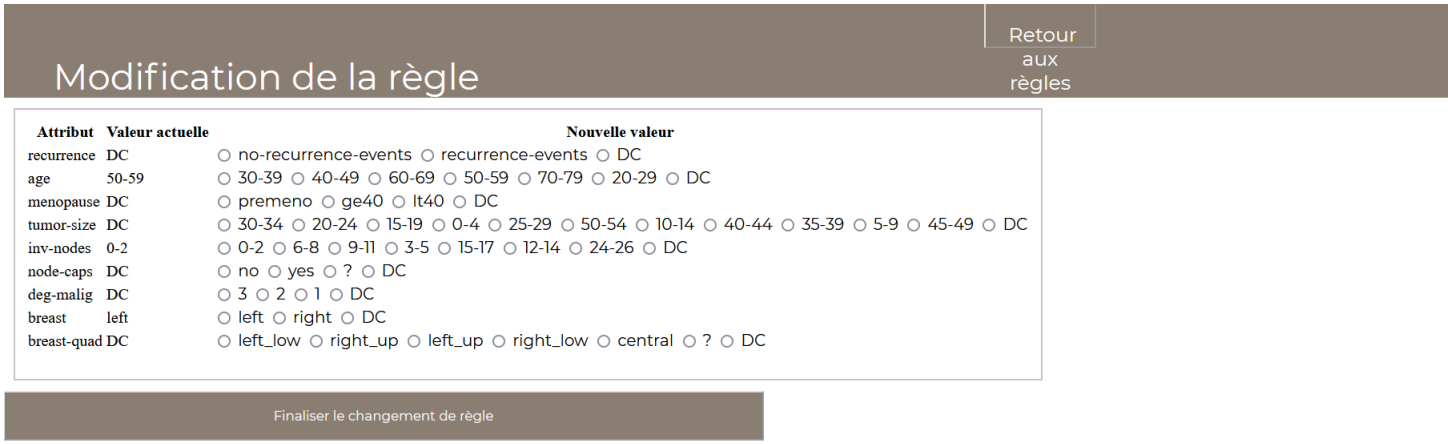


Figure 6.5: Interface utilisateur : modification d'une règle

6.3 Conclusion

Ce chapitre montre à l'aide de screenshots ce à quoi l'interface utilisateur ressemble et comment elle fonctionne. Bien qu'une amélioration du design est nécessaire, le logiciel est fonctionnel et permet d'utiliser la méthodologie présentée dans le chapitre précédent. Grâce à ce logiciel, des expérimentations peuvent avoir lieu pour analyser l'intérêt de la méthodologie. Ces tests sur les différentes opérations réalisables sur l'ensemble de règles sont disponibles dans le chapitre suivant.

Chapitre 7

Expérimentation

L'interface utilisateur étant présentée, ce chapitre développe quelques expérimentations basées sur les différentes modifications applicables sur l'ensemble de règles. Ces tests ont pour but de valider les différentes opérations sur l'ensemble de règles avec plusieurs exemples. La performance des arbres de décision générés est calculée pour donner une idée de son amélioration ou de sa baisse après l'opération sur l'ensemble de règles. Toutefois, ces expérimentations se déroulent sans expert du domaine et les résultats obtenus doivent être analysés avec précaution. Une hausse en performance n'est pas toujours bénéfique pour un modèle, comme lorsqu'une classe est plus importante à prédire qu'une autre.

Ce chapitre passe en revue les cinq différentes opérations possibles sur les règles : la suppression d'une règle, l'ajout d'une règle, la suppression d'une condition dans une règle, l'ajout d'une condition et le changement de classe d'une règle. Pour cela, trois datasets publics ayant en commun le domaine médical sont utilisés.

7.1 Suppression d'une règle

Pour expérimenter la suppression d'une règle, le dataset public *breast cancer* est utilisé. Il contient 286 instances et 10 features différentes. Les patientes sont répertoriées comme étant malades ou saines sous l'attribut *irradiat*. Les 10 attributs sont catégoriels et RIPPERk peut donc directement être appliqué sur le dataset d'entraînement correspondant à 70% du dataset total. L'ensemble de règles en résultant est repris dans le tableau 7.1. Cet ensemble de règles peut être ensuite transformé en un arbre de décision de la façon présentée au chapitre 5. L'arbre résultant de l'ensemble

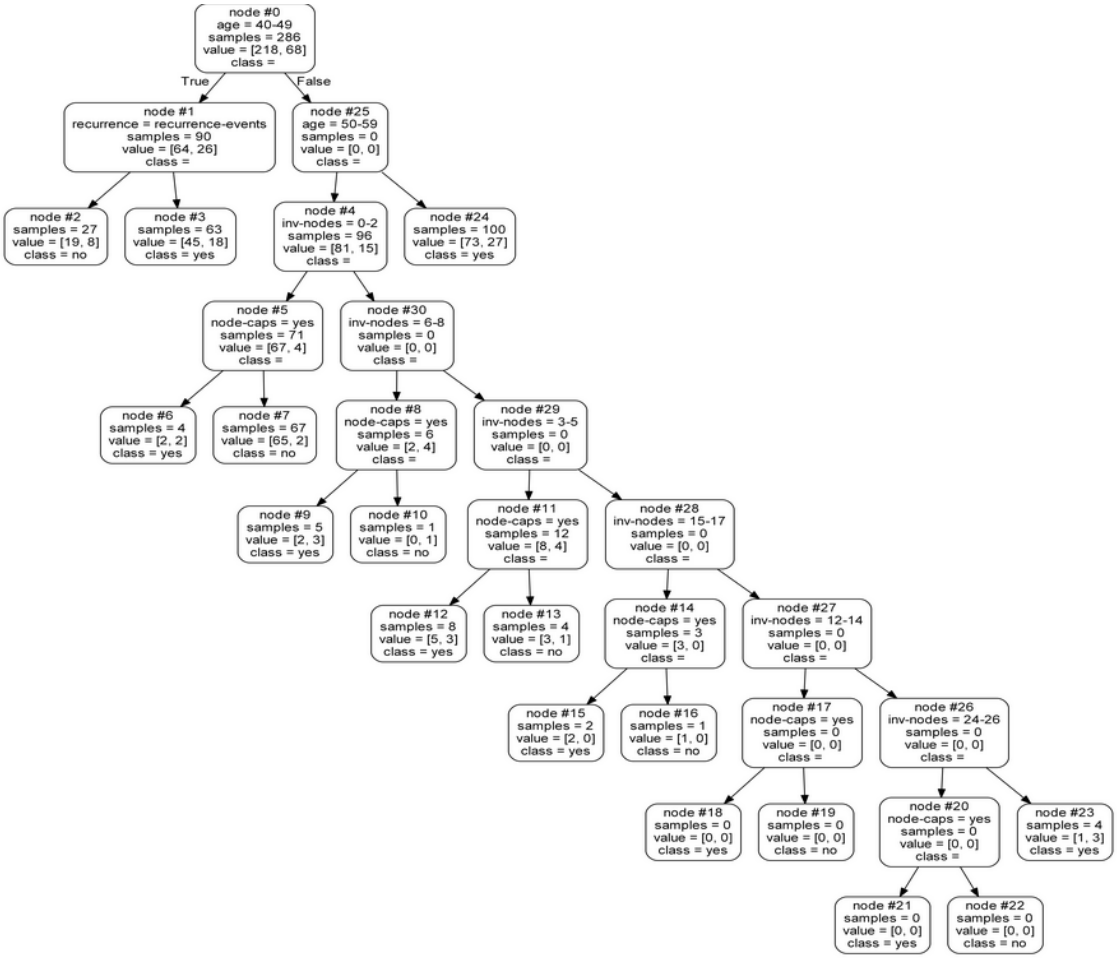
Règle 1	IF $recurrence = no - recurrence - events$ AND $age = 50 - 59$ AND $node - caps = no$ THEN No
Règle 2	IF $recurrence = recurrence - events$ AND $age = 40 - 49$ THEN No
Règle 3	IF $node - caps = yes$ AND $deg - malig = 3$ AND $breast - quad = left_low$ THEN Yes
Règle 4	IF $recurrence = no - recurrence - events$ AND $age = 40 - 49$ AND $node - caps = yes$ THEN Yes
Règle 5	IF $tumor - size = 30 - 34$ and $inv - nodes = 9 - 11$ THEN Yes

Table 7.1: Ruleset induit par RIPPERk

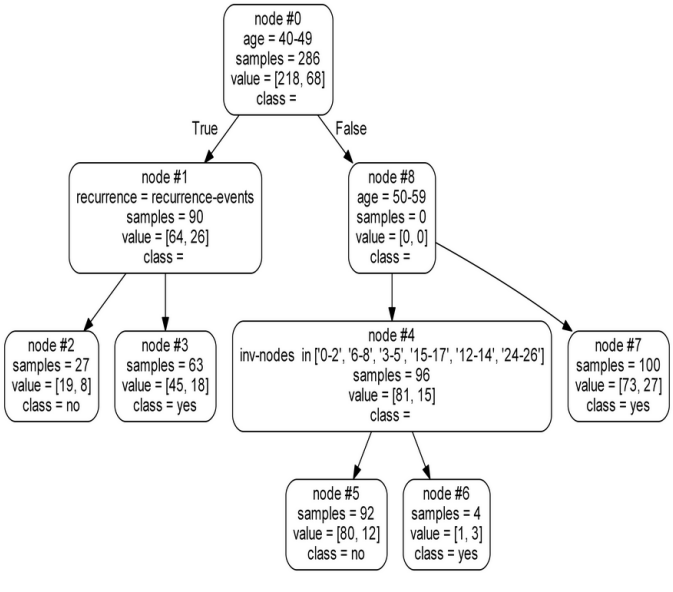
de règles ci-dessus est repris à la figure 7.1a. Il atteint une performance de 50.5% d'instances correctement classifiées pour le dataset d'entraînement et 52.33% pour le dataset de test. Toutefois cette performance est à relativiser étant donné que la répartition des instances dans les classes est disproportionnée. La performance balancée ou *balanced accuracy* peut être plus appropriée dans cet exemple.

Pour déterminer quelle règle supprimer, leur performance a été déterminée en fonction du pourcentage d'instances correctement classifiées sur toutes les instances couvertes. La règle 3, n'obtenant que 53.85% de performance, est la règle la moins optimale et est donc supprimée. L'arbre de décision résultant du nouvel ensemble de règles est présenté à la figure 7.1b. La suppression de la règle résulte en une taille d'arbre beaucoup plus réduite ce qui améliore grandement son explicabilité et sa lisibilité. De plus, l'arbre suite à la modification de l'ensemble de règles est également légèrement plus performant que l'arbre original. En effet, 50.5% des instances du dataset d'entraînement et 53.49% des instances du dataset de test sont classifiées correctement par ce nouvel arbre.

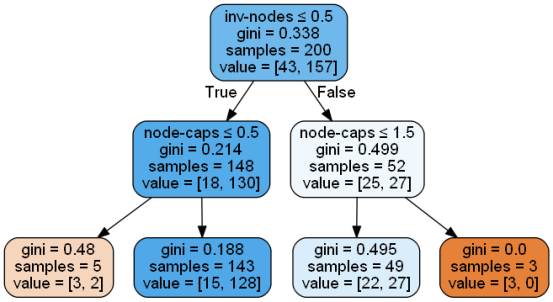
L'arbre de décision généré par ID3 est nettement meilleur en terme de performance que les deux arbres précédents. Il atteint 80.5% de performance sur le dataset d'entraînement et 73.26% sur le dataset de test. Cet arbre est assez restreint car il ne contient que trois noeuds et quatre feuilles mais parvient à sélectionner de meilleurs attributs pour ces noeuds que les deux arbres présentés précédemment. Il est visible à la figure 7.1c.



(a) Arbre de décision généré par l'ensemble de règles



(b) Arbre de décision après la suppression de la règle



(c) Arbre de décision généré par ID3

Figure 7.1: Arbres générés sur base du dataset *breast-cancer*

7.2 Suppression d'une condition

L'expérimentation de la suppression d'une condition est réalisée sur le dataset public *Haberman's survival* qui contient 306 instances. Il répertorie les patients en fonction de leur décès ou leur survie dans les cinq ans après une opération. Trois attributs caractérisent ces patients : leur âge lors de l'opération, l'année de l'opération et le nombre de ganglions axillaires positifs détectés. Pour pouvoir utiliser RIPPERk et ID3, il est nécessaire de discrétiser les trois features. Elles sont donc divisées chacune en trois bins comme cela avait déjà été réalisé dans l'article (Nanfack 2020). L'ensemble de règles généré par RIPPERk est disponible dans le tableau 7.2. En ne modifiant au-

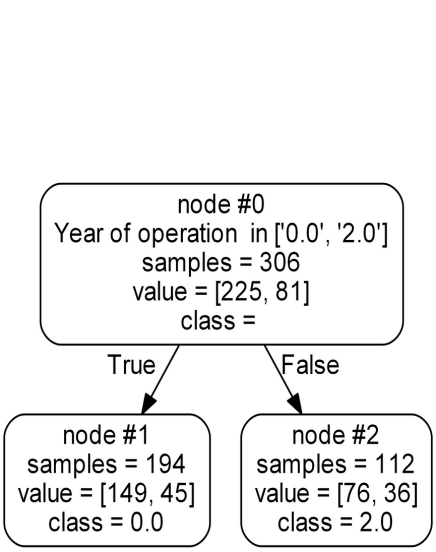
Règle 1	IF $Age = 48 - 65$ AND $YearOfOperation >= 66$ AND $NumberOfNodes = 18 - 34$ THEN 1
Règle 2	IF $Age <= 47$ AND $YearOfOperation <= 61$ AND $NumberOfNodes >= 17$ THEN 1
Règle 3	IF $Age = 48 - 65$ AND $YearOfOperation = 62 - 65$ AND $NumberOfNodes = 18 - 34$ THEN 2

Table 7.2: Ruleset induit par RIPPERk

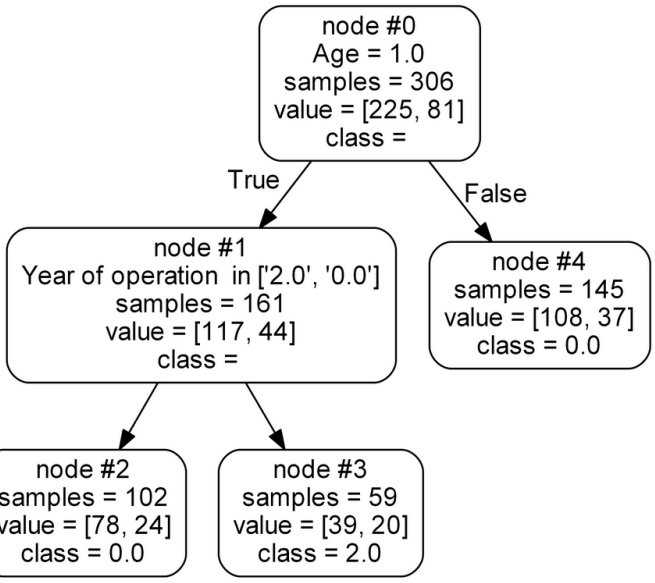
cune règle de décision, l'arbre présent à la figure 7.2a est généré. Il ne contient qu'un seul noeud et deux feuilles. L'arbre obtenu peut être à première vue surprenant en comparaison avec l'ensemble de règles desquelles il découle. En effet, un seul attribut est sélectionné pour construire l'arbre alors que chacune des trois règles de décision contient trois attributs. Toutefois, AQDT dans son implémentation s'arrête dès que toutes les feuilles contiennent des règles de la même classe (voir section 3.2.1). Dans cet exemple, après avoir sélectionné l'attribut *YearOfOperation*, les règles 1 et 2 suivent la branche de gauche tandis que la règle 3 suit la branche de droite. Dans la première feuille, les règles 1 et 2 appartiennent toutes deux à la même classe. Dans la deuxième feuille, seule la règle 3 et donc la classe 2 sont présentes. AQDT peut donc déjà arrêter la construction de l'arbre. En fin de compte, l'arbre obtenu obtient respectivement une performance de 59.81% et 61.96% sur les datasets d'entraînement et de test.

Comme toutes les valeurs de l'attribut *YearOfOperation* sont différentes pour chacune des règles, il est intéressant de modifier légèrement une des trois règles pour que l'arbre soit légèrement plus complexe. La condition $YearOfOperation <= 61$ dans la règle 2 est donc supprimée et un arbre de décision est généré avec l'aide de l'interface présentée dans le chapitre 6. Cet arbre apparaît à la figure 7.2b. Il est légèrement plus grand que l'arbre original car il a deux noeuds et trois feuilles. Sa performance est également meilleure que celle de son homologue. 66.36% et 69.57% sont ses pourcentages d'instances correctement classifiées pour le dataset d'entraînement et celui de

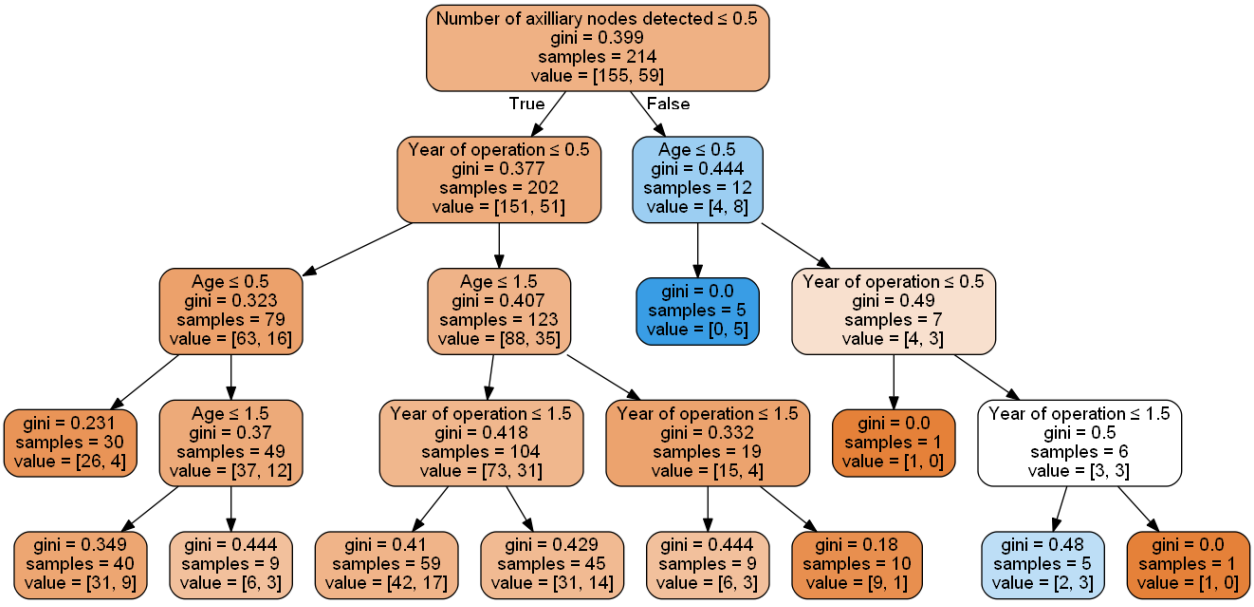
test. Toutefois, ces deux arbres sont moins performants que celui induit par ID3 (voir figure 7.2c) qui obtient 75 et 77% de performance sur les datasets d'entraînement et de test. La différence de complexité entre l'arbre généré par ID3 et ceux générés par AQDT est également à mettre en avant.



(a) Arbre de décision généré par l'ensemble de règles



(b) Arbre de décision après la suppression de la condition



(c) Arbre de décision généré par ID3

Figure 7.2: Arbres générés sur base du dataset *Haberman's survival*

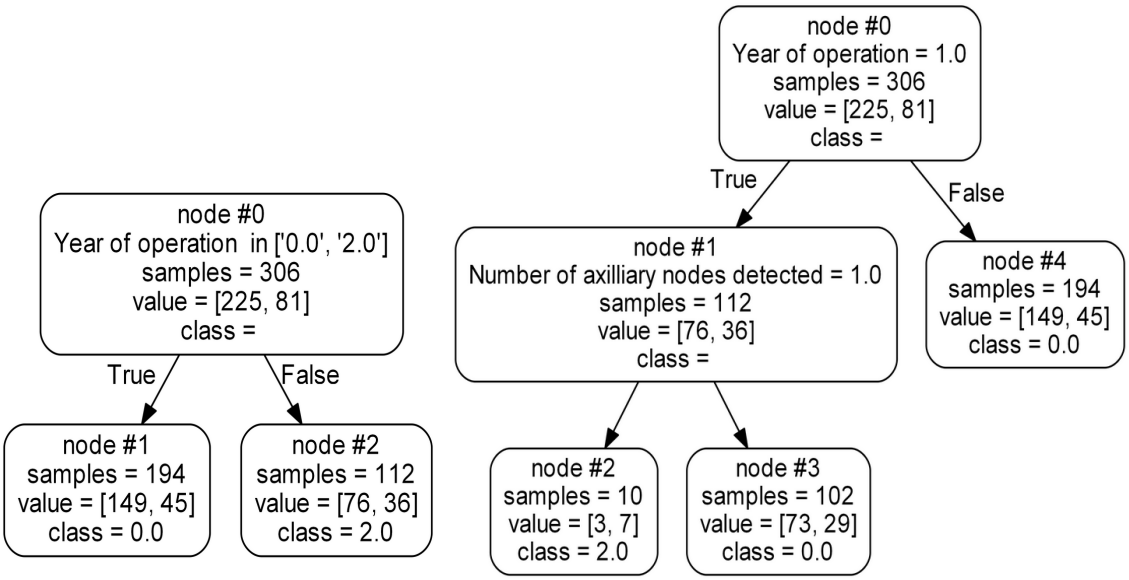
7.3 Ajout d'une règle

L'expérimentation de l'ajout d'une règle se déroule avec le même dataset que celui de la section précédente (7.2). L'ensemble de règles est donc le même que celui du tableau 7.2. L'arbre de décision obtenu avant modification de l'ensemble de règles est donc également le même que celui présenté à la figure 7.2a. Il est repris à la figure 7.3a. Enfin, l'arbre généré par ID3 est également identique à celui de la figure 7.2c car se basant sur les mêmes données.

Dans l'arbre original, la feuille 2 n'est peut-être pas assez développée et ajouter une règle pourrait permettre de subdiviser cette feuille. En ajoutant la règle

$$\text{IF } YearOfOperation = 62 - 65 \text{ AND } NumberOfNodes >= 35 \text{ THEN } 1$$

, l'ensemble de règles induit l'arbre de décision présent à la figure 7.3b. Cet arbre atteint une performance de 75.23% pour le dataset d'entraînement et de 73.91% pour celui de test. Toutefois, il est intéressant de noter qu'avec cet arbre, seulement sept patients de la classe 2 sont bien classifiés. L'arbre original, quant à lui, se trompe plus souvent mais parvient à trouver 36 patients sur les 81 patients de la classe 2.



(a) Arbre de décision généré par l'ensemble de règles

(b) Arbre de décision après l'ajout de la règle

Figure 7.3: Arbres générés sur base du dataset *Haberman's survival*

7.4 Ajout d'une condition

Pour expérimenter l'ajout d'une condition dans une règle, un troisième et dernier dataset public est utilisé. Celui-ci s'appelle *post operative patient* et contient 90 patients classés en fonction de l'endroit où ils sont envoyés après une opération. Certains sont internés en soins intensifs (I), d'autres restent en observation à l'hôpital (A) et les derniers peuvent rentrer chez eux (S).

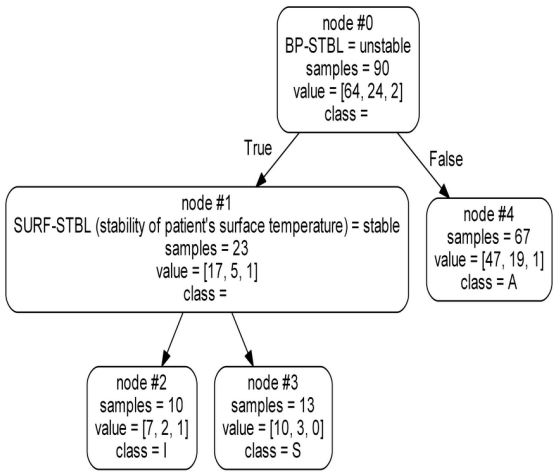
RIPPERk induit l'ensemble de règles visible au tableau 7.3 grâce au données d'entraînement du dataset. L'arbre de décision généré par AQDT sur base de ces règles est quant à lui présent à la figure 7.4a. Il contient deux noeuds et trois feuilles appartenant à chacune des trois classes. Ses performances sur les datasets d'entraînement et de test sont respectivement de 55.56% et 59.26%.

Règle 1	IF $L-O2 = good$ AND $BP-STBL = mod-stable$ THEN A
Règle 2	IF $L-SURF = low$ AND $BP-STBL = unstable$ THEN S
Règle 3	IF $L-SURF = low$ AND $SURF-STBL = stable$ AND $BP-STBL = unstable$ THEN I

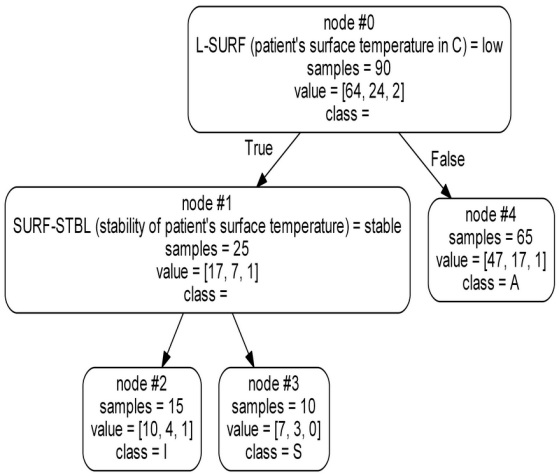
Table 7.3: Ruleset induit par RIPPERk

Une condition intéressante à ajouter est la suivante : $L-SURF = high$ dans la règle 1. En effet, beaucoup d'instances dans le dataset ayant la valeur $high$ pour l'attribut $L-SURF$ appartiennent à la classe A. En réalisant cette modification, l'arbre de décision à la figure 7.4b est obtenu. Cet arbre est sensiblement similaire à l'arbre original et obtient d'ailleurs exactement les mêmes scores de performances que celui-ci. La grande différence est l'attribut sélectionné dans le premier noeud. Avant la modification $BP-STBL$ (la stabilité de la pression sanguine) était l'attribut sélectionné. Après celle-ci, c'est l'attribut $L-SURF$ (la température de la surface de la peau du patient) qui est privilégié. Ce petit changement peut avoir une utilité lorsque certains attributs sont plus difficiles à obtenir ou plus coûteux que d'autres. Dans cet exemple précis, La température du patient peut être plus facile à mesurer que la stabilité de sa pression sanguine.

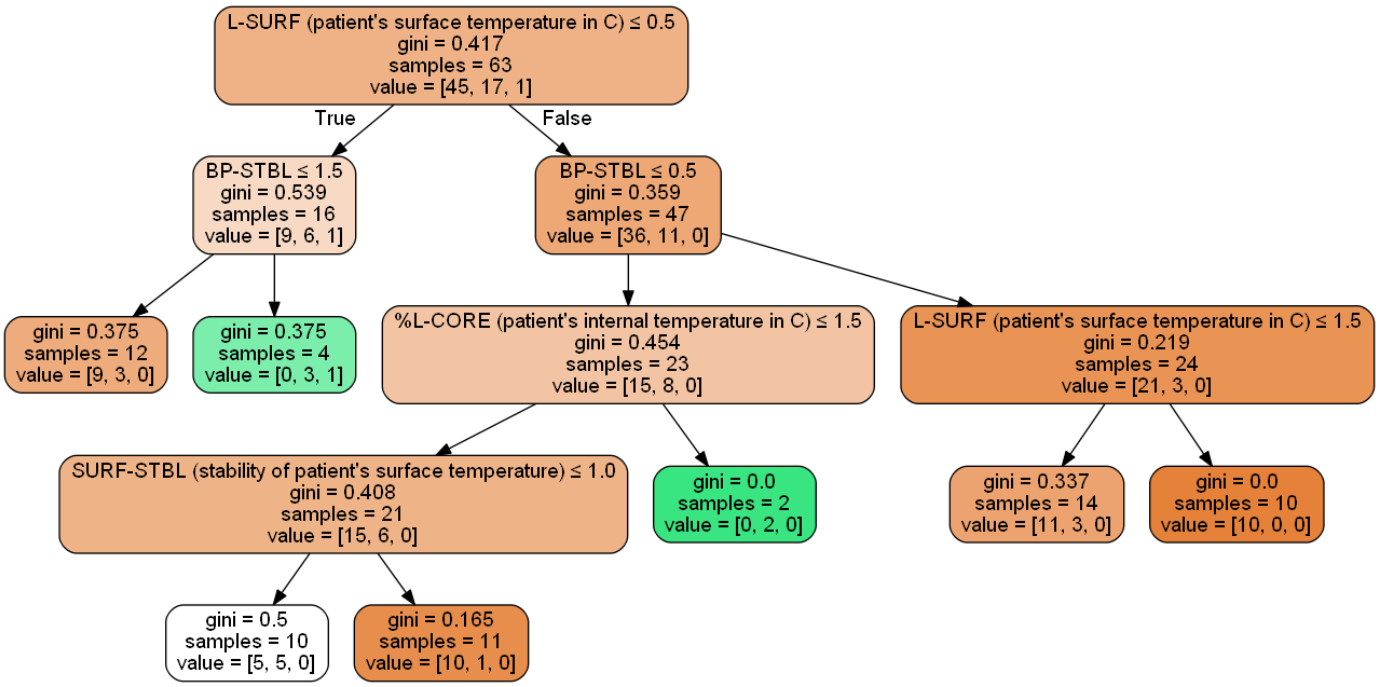
Enfin, l'arbre de décision généré par ID3 est plus grand que ceux réalisés avec AQDT. Il a six noeuds et sept feuilles et obtient 79.36% et 59.26% de performance sur les datasets d'entraînement et de test. Le premier attribut sélectionné est également le même que pour l'arbre induit par les règles modifiées : $L-SURF$.



(a) Arbre de décision généré par l'ensemble de règles



(b) Arbre de décision après l'ajout de la condition



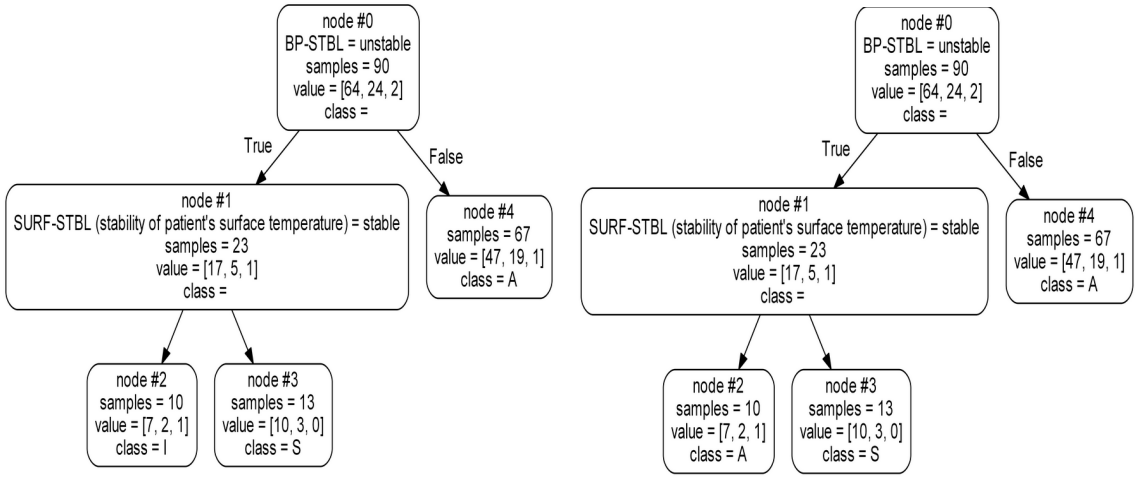
(c) Arbre de décision généré par ID3

Figure 7.4: Arbres générés sur base du dataset *Haberman's survival*

7.5 Changement de classe

Le changement de classe pour une règle de décision est la dernière opération expérimentée dans ce chapitre. Comme pour l'ajout de condition dans la section précédente, le dataset public *post operative patient* est utilisé. L'ensemble de règles dérivé par RIPPERk et l'arbre de décision généré sont donc identiques à ceux de la section précédente (voir tableau 7.3 et figure 7.4a).

Le changement de classe s'opère sur la règle 3. Au lieu de pointer vers la classe I, la règle 3 est modifiée pour appartenir à la classe A. En effet, la majorité des instances du dataset d'entraînement appartenant à la règle 3 ont pour attribut cible la classe A. L'arbre de décision résultant de ce changement est visible à la figure 7.5b. Il est en tout point identique à l'arbre original à l'exception de la classe prédite par la feuille 2. En effet, la classe prédite est A dans le nouvel arbre par rapport à la classe I dans le premier arbre généré. L'amélioration de performance résultant de cette modification est significative. En effet, 63.49% et 62.96% des instances des datasets d'entraînement et de test sont correctement classifiées par cet arbre modifié. Toutefois, plus aucune feuille ne prédit la classe I. Concrètement, cela signifie qu'aucun futur patient ne sera pronostiqué comme ayant besoin des soins intensifs après l'opération. Ce dernier type de modification est toutefois assez fort car c'est aller complètement à l'encontre du modèle de machine learning. L'opération de changement de classe convient donc d'être utilisée dans des cas très spécifiques.



(a) Arbre de décision généré par l'ensemble de règles

(b) Arbre de décision après le changement de classe

Figure 7.5: Arbres de décision générés sur base du dataset *post operative patient*

7.6 Conclusion

Ce chapitre montre qu'il est possible de réaliser cinq opérations différentes sur n'importe quel ensemble de règles de décision. Ces modifications ont pour effet de modifier de façon plus ou moins importante l'arbre de décision final. Cela impacte la performance du modèle mais également sa complexité, son explicabilité et sa lisibilité. Dans la section 7.1, l'arbre de décision après la

suppression de la règle est nettement plus lisible. Pour conclure ce chapitre, les expérimentations réalisées montrent qu'il existe certains avantages à réaliser des modifications sur l'ensemble de règles tant au niveau de la performance que de l'explicabilité du modèle.

Chapitre 8

Améliorations possibles pour l'ajout de contraintes

Dans le chapitre précédent, plusieurs expérimentations ont pu être réalisées sur les différentes opérations permettant d'interagir avec l'ensemble de règles. Toutefois, toutes les opérations réalisées ne représentent pas toutes les contraintes possiblement implémentables et utiles à un expert du domaine. Dans l'article *Learning Customised Decision Trees for Domain-knowledge Constraint* (Nanfack 2020), plusieurs types de contraintes sont présentées. Par exemple, une hiérarchie sur les attributs peut être établie ou un attribut spécifique doit pouvoir apparaître obligatoirement dans l'arbre de décision. Ce chapitre a pour but de présenter des pistes pour une éventuelle future implémentation de certains types de contraintes dans la méthodologie présentée.

8.1 Premier type de contrainte : exclusion sur une branche

La premier type de contrainte analysé est l'exclusion sur une branche. Cela a pour but d'interdire à un attribut d'être présent sur une branche de l'arbre de décision si un autre attribut spécifique est déjà présent sur cette même branche. Si chaque chemin décisionnel de l'arbre est transformé en une règle de décision telle que

$$\text{IF } c_1 \wedge c_2 ... \wedge c_i ... \wedge c_k \text{ THEN } y = C$$

avec k le nombre d'attributs, C la classe prédite par la règle, et c_i une condition telle que

$$c_i \equiv \mathrm{A}_i \in \mathrm{V}_i$$

```
Procedure ExclusionOnRuleset (Ruleset,FirstAttribute, SecondAttribute):
For Rule in Ruleset.rules:
      If FirstAttribute in Rule.attributes and SecondAttribute in Rule.attributes:
            FirstAlternativeRule = Rule.attributes / {SecondAttribute}
            SecondAlternativeRule = Rule.attributes / {FirstAttribute}
            ScoreFirstAttribute = Performance(FirstAlternativeRule)
            ScoreSecondAttribute = Performance(SecondAlternativeRule)
            If ScoreFirstAttribute > ScoreSecondAttribute :
                  NewRule = FirstAlternativeRule
            Else:
                  NewRule = SecondAlternativeRule
            Ruleset.remove_rule(Rule)
            Ruleset.add_rule(NewRule)
Return (Ruleset)
```

Figure 8.1: Algorithme d'implémentation de la contrainte d'exclusion sur une branche

avec A_i le ième attribut et V_i l'ensemble des valeurs possibles de l'attribut en question; alors la contrainte présentée ci-dessus par

$$ExclusionOnBranch(S_i, A_a, A_b) \equiv \forall R_i \in S_i : \neg(A_a \in R_i \wedge A_b \in R_i) \tag{8.1}$$

avec S_i l'ensemble des règles décisionnelles correspondant aux branches de l'arbre.

8.1.1 Implémentation de la contrainte

Les contraintes étant implémentées lors de la modification des règles et non lors de la génération de l'arbre avec l'algorithme AQDT, l'implémentation de la contrainte 8.1 n'est pas trivial. Toutes les règles présentes dans l'ensemble de règles à modifier ne sont pas présentes sous forme de branche dans l'arbre de décision final. Toutefois, la contrainte d'exclusion sur une branche est transposée sur l'ensemble de règles pour tenter de forcer la contrainte sur l'arbre de décision également. En résumé, aucune règle de l'ensemble de décision ne permettra de contenir les deux attributs A_a et A_b. Pour chaque règle contenant les deux attributs, deux règles alternatives sont construites en supprimant un des deux attributs. Les deux règles sont ensuite évaluées en terme de performance en fonction du pourcentage d'instances correctement classifiées par la règle. La meilleure des deux alternatives remplace ensuite la règle initiale. L'algorithme représentant l'implémentation de ce type de contrainte est présente à la figure 8.1.

8.2 Deuxième contrainte : hiérarchie d'attributs

La deuxième contrainte analysée dans ce chapitre est appelée la hiérarchie d'attributs. Cette contrainte impose à une feature de ne pas apparaître avant une autre feature dans un chemin décisionnel. En notant tous les chemins décisionnels sous forme de règles de décision comme pour la contrainte précédente, la contrainte se formalise sous la forme de l'équation

$$AttributeHierarchy(S_i, A_a, A_b) \equiv \forall R_i \in S_i : \neg((A_a \in R_i \wedge A_b \in R_i) \wedge (\exists j, k : A_a \in c_j \wedge A_b \in c_k \wedge k < j)). \tag{8.2}$$

Pour toute règle de décision dérivée de l'arbre de décision, si les attributs apparaissent en même temps dans cette règle, alors l'attribut A_b n'apparaît jamais avant l'attribut A_a.

8.2.1 Implémentation de la contrainte

Pour implémenter cette contrainte, la solution proposée dans cette section est de se baser sur l'algorithme AQDT. Pour qu'un attribut soit sélectionné avant un autre, il faudrait qu'AQDT le considère comme un meilleur choix que son alternative. Pour cela, la disjointeté de l'attribut est le critère majeur qui entre en compte. Il serait donc judicieux d'opérer une modification de la disjointeté de l'un ou l'autre des deux attributs en question. Cette modification peut se dérouler de deux manières différentes.

Tout d'abord, elle peut se faire via un changement de l'ensemble de règles. Chaque opération sur une règle à laquelle appartient un attribut a pour effet de modifier la disjointeté d'une certaine manière, que ce soit l'augmenter ou la diminuer. La seconde méthode possible pour modifier la disjointeté d'un attribut est de lui augmenter artificiellement sa dsijointeté d'un montant spécifique sans changer pour autant l'ensemble de règles. Cela revient donc à donner à AQDT une valeur augmentée ou diminuée de la disjointeté réelle de l'attribut. Chacune des deux solutions a ses propres questions à répondre. Pour la première solution, il faut déterminer *quelle opération réaliser dans quelle situation.* Pour la deuxième solution, il est nécessaire de déterminer quel est le montant de disjointeté à ajouter ou diminuer à l'attribut.

Des tests plus approfondis doivent être réalisés pour déterminer quelle solution privilégier. Pour cela, la performance du modèle doit être prise en compte mais également le respect ou non de la contrainte de départ. En effet, le respect de ce type de contrainte ne peut pas être garanti avec la

solution présentée.

8.3 Troisième contrainte : attribut devant être sélectionné

Enfin, un troisième type de contrainte est analysé dans ce chapitre. Cette contrainte ordonne à un des attributs d'absolument apparaître dans l'arbre de décision final, que ce soit au sommet ou au centre de l'arbre. Toujours en reprenant les différents chemins décisionnels de l'arbre en question comme des règles de décision, cette contrainte se formalise avec l'équation

$$ImperativeAttribute(S_i, A_a) \equiv \exists R_i \in S_i : A \in R_i. \tag{8.3}$$

8.3.1 Implémentation de la contrainte

L'implémentation de la contrainte est similaire à celle de la deuxième contrainte de hiérarchie entre les attributs. En effet, pour promouvoir un attribut en particulier afin qu'il soit sélectionné dans l'arbre, sa disjointeté peut être augmentée. De manière équivalente à la discussion entamée dans la section précédente, la disjointeté peut être augmentée en modifiant l'ensemble de règles ou *artificiellement* sans changer celui-ci. L'implémentation de cette contrainte nécessite également des expérimentations plus poussées pour déterminer la solution optimale.

8.4 Conclusion

Ce chapitre a uniquement pour but de susciter une réflexion future sur l'implémentation de contraintes de plus haut niveau dans l'induction d'arbres de décision. Une version idéale du logiciel présenté dans le chapitre 6 permettrait à l'utilisateur d'insérer ses propres contraintes haut niveau telles que celles présentées dans ce chapitre. Ces contraintes devraient idéalement être respectées dans tous les cas sans dégrader excessivement la performance du modèle.

Chapitre 9

Conclusion et améliorations

Après cette expérimentation proposée dans le chapitre précédent, toutes les grandes parties de ce mémoire ont été abordées. Après avoir introduit le sujet, les arbres de décision et les règles décisionnelles ont été présentées (voir chapitre 2). Une revue de la littérature a été réalisée dans le chapitre 3 puis l'avis de plusieurs experts a été répertorié dans le chapitre 4. Enfin, la méthodologie a été présentée (chapitre 5), implémentée (chapitre 6) et expérimentée (chapitre 7). Ce dernier chapitre conclut ce travail en abordant les différents résultats réalisés ainsi que les limites et les possibles améliorations de la méthodologie et du logiciel.

9.1 Résultats et objectifs atteints

Tout au long de ce travail, beaucoup d'obstacles ont été surmontés pour atteindre ce résultat final. Certains objectifs sont d'ailleurs à souligner. Premièrement, le logiciel développé est opérationnel et permet d'expérimenter la méthodologie présentée. Le développement de ce logiciel a nécessité d'intégrer une librairie pour l'algorithme RIPPERk, d'implémenter de A à Z les algorithmes RBDT et AQDT et de réaliser une interface utilisateur pour interagir avec le code en *backend*. Ce logiciel peut être vu comme l'aboutissement de ce travail.

Deuxièmement, les interviews réalisées ont permis d'avoir plusieurs avis extérieurs sur la méthodologie développée. Les experts ont pointé plusieurs aspects intéressants sur lesquels des améliorations peuvent être engagées et ont montré leur enthousiasme pour l'approche présentée. Ces avis extérieurs généralement positifs confortent évidemment sur l'utilité et l'application future de la méthodologie.

Enfin, l'approche en elle-même est un résultat à souligner dans cette conclusion. En effet, cette méthode développée atteint l'objectif de départ de ce travail : permettre à un expert de domaine d'apporter son expérience à un modèle de machine learning. Elle a l'avantage de permettre à l'expert de découvrir ce que le modèle génère comme règles de décision avant d'incorporer ses propres connaissances et son expérience.

9.2 Améliorations et limites

Malgré les bons résultats obtenus, certaines améliorations sont toujours possibles. Certaines ont, par exemple, été pointées par les experts lors des interviews. Tout d'abord, la méthodologie présentée ne permet d'intégrer que des features ayant des valeurs catégorielles. Cela restreint évidemment beaucoup la méthode étant donné qu'énormément d'attributs ne sont pas catégoriels dans la plupart des projets business actuels. Toutefois, cette contrainte peut être légèrement contournée en discrétisant les attributs continus.

De plus, la méthodologie développée ne permet pas d'ajouter tous les différents types de contraintes. Un expert du domaine ne peut qu'ajouter, supprimer ou modifier des règles de décision. Cependant, plusieurs contraintes intéressantes ne peuvent pas être trivialement ajoutées par l'expert du domaine. Par exemple, une hiérarchie entre les instances ou un coût maximal à ne pas dépasser pour la prédiction sont des contraintes ne pouvant pas être modélisées par l'approche présentée.

Enfin, le logiciel, bien qu'opérationnel, peut également être grandement amélioré. En effet, l'aspect esthétique du logiciel pourrait être revu et amélioré. D'autres fonctionnalités peuvent aussi être ajoutées à la version actuelle du logiciel. Un exemple pourrait être une introduction optionnelle du coût des différentes variables par l'expert du domaine.

Annexes

Annexe A

Questions interview

A.1 Questions contraintes techniques

- Tout d'abord, est-ce que vous pourriez expliquer brièvement en quoi consiste votre métier, et à quel point vous utilisez le machine learning dans la vie de tous les jours ?
- Si oui, pourriez-vous expliquer quelles tâches vous réalisez (régression, classification, ...) ? Sur quels types de données ?
- Vous est-il déjà arrivé de définir des contraintes techniques (mémoire, vitesse de calcul, ...) lors du développement d'un modèle ? Si oui, comment avez-vous fait pour définir ces contraintes ?
- En ce qui concerne les contraintes techniques, à quel point considérez-vous utile/est-ce fréquent de définir de telles contraintes lorsque vous développez un modèle de ML ?

A.2 Questions contraintes opérationnelles

- Même question que pour les contraintes techniques, vous est-il déjà arrivé de définir des contraintes métiers (axées plus sur le dataset) lors du développement d'un modèle ? Si oui, comment avez-vous fait pour définir ces contraintes ?
- A quel point considérez-vous utile/est-ce fréquent de définir des contraintes opérationnelles pour générer un modèle ML ?

- Dans votre expérience personnelle, quel type de contraintes (coût, instances plus importantes que d'autre, connaissance ou restriction technique de machine) avez-vous le plus souvent rencontré ?
- Plus particulièrement, avez-vous déjà travaillé avec les arbres de décision ? Si oui, avez-vous déjà expérimenté des problèmes avec le fonctionnement de l'arbre ? Comment avez-vous fait pour régler ce problème ? (en ajoutant des contraintes ?)

Annexe B

Mise en situation

B.1 1ère méthode : arbre de décision

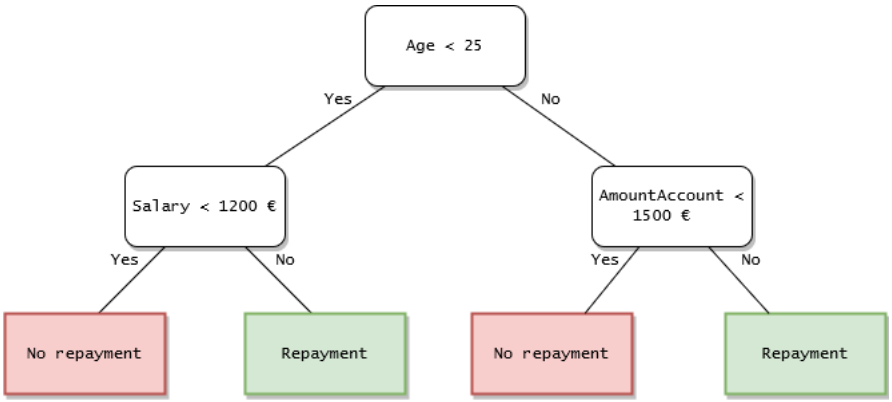


Figure B.1: Exemple fictif d'arbre de décision généré par ID3

B.2 2ème méthode : ensemble de règles original

Conditions	Classe
IF *Age <25* AND *Salary <1200€*	No repayment
IF *Age <25* AND *Salary >1200€*	Repayment
IF *Age >25* AND *AmountAccount <1500€*	No repayment
IF *Age >25* AND *AmountAccount >1500€*	Repayment

Figure B.2: Ensemble de règles fictif généré par AQDT

B.3 2ème méthode : ensemble de règles modifié

Conditions	Classe
IF *Salary <1200€*	No repayment
IF *Age <25* AND *Salary >1200€*	Repayment
IF *Age >25* AND *AmountAccount <1500€*	No repayment
IF *Age >25* AND *AmountAccount >1500€*	Repayment

Figure B.3: Ensemble de règles fictif modifié par le banquier

B.4 2ème méthode : arbre de décision final

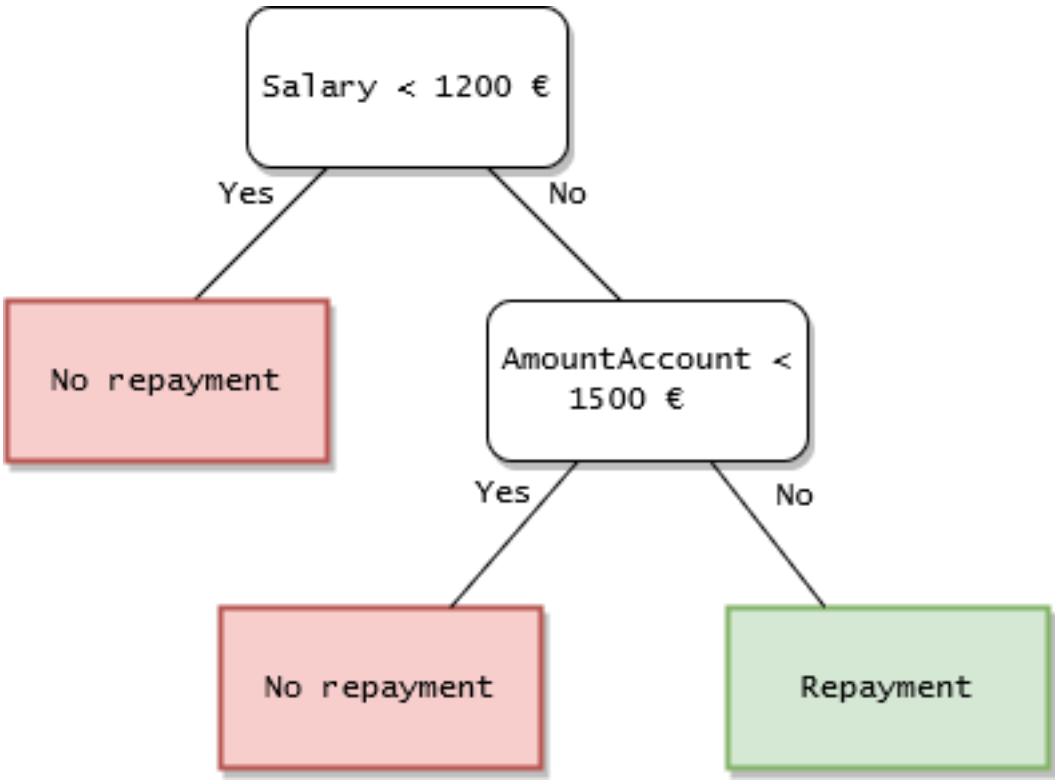


Figure B.4: Exemple fictif d'arbre de décision généré par RIPPERk

B.5 Questions sur la mise en situation

- Trouvez-vous utile qu'un expert du métier puisse modifier un modèle de machine learning ou devrait-il toujours se fier aux résultats du modèle ?

- Si vous aviez dû inclure les connaissances du banquier, comment auriez-vous fait ? Pensez-vous qu'en passant par des règles de décision, il est possible de modéliser toutes les connaissances et les contraintes du banquier ?
- A quel point jugez-vous utile un outil de ce type permettant de modifier des règles de décision (+ ajouter et supprimer) en vue de réaliser un arbre de décision ?
- Dans votre métier, avez-vous déjà connu des situations où un tel outil aurait été utile et intéressant ?

Bibliographie

[ATS09] Amany Abdelhalim, Issa Traore, and Bassam Sayed. "RBDT-1: a new rule-based decision tree generation technique". In: *International Workshop on Rules and Rule Markup Languages for the Semantic Web*. Springer. 2009, pp. 108–121.

[BH90] Giulia Bagallo and David Haussler. "Boolean feature discovery in empirical learning". In: *Machine learning* 5.1 (1990), pp. 71–99.

[Bon17] Giuseppe Bonaccorso. *Machine learning algorithms*. Packt Publishing Ltd, 2017.

[BP91] Clifford A Brunk and Michael J Pazzani. "An investigation of noise-tolerant relational concept learning algorithms". In: *Machine Learning Proceedings 1991*. Elsevier, 1991, pp. 389–393.

[Coh93] William W. Cohen. "Efficient Pruning Methods for Separate-and-Conquer Rule Learning Systems". In: *IJCAI*. 1993.

[Coh95] William W Cohen. "Fast Effective Rule Induction". In: (1995).

[Fré20] Benoît Frénay. "Notes du cours IDASM102". In: (2020).

[FW94] Johannes Fürnkranz and Gerhard Widmer. "Incremental reduced error pruning". In: *Machine Learning Proceedings 1994*. Elsevier, 1994, pp. 70–77.

[IM93] Ibrahim F. Imam and Ryszard S. Michalski. "Learning decision trees from decision rules: a method and initial results from a comparative study". In: *Journal of Intelligent Information Systems* 2.3 (1993), pp. 279–304.

[Meh+21] Ninareh Mehrabi et al. "A survey on bias and fairness in machine learning". In: *ACM Computing Surveys (CSUR)* 54.6 (2021), pp. 1–35.

[Min89] John Mingers. "An empirical comparison of pruning methods for decision tree induction". In: *Machine learning* 4.2 (1989), pp. 227–243.

[Mit+97] Tom M Mitchell et al. “Machine learning”. In: (1997).

[Nan20] Géraldin Nanfack. “Learning Customised Decision Trees for Domain-knowledge Constraint”. In: (2020).

[Qui14] J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[Qui83] J Ross Quinlan. “Learning efficient classification procedures and their application to chess end games”. In: *Machine learning*. Springer, 1983, pp. 463–482.

[Qui87] J Ross Quinlan. “Generating production rules from decision trees.” In: *ijcai*. Vol. 87. Citeseer. 1987, pp. 304–307.

[Qui95] J Ross Quinlan. “MDL and categorical theories (continued)”. In: *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 464–470.

[Sam59] Arthur L Samuel. “Some Studies in Machine Learning Using the Game of Checkers”. In: (1959).

[Sch93] Cullen Schaffer. “Overfitting avoidance as bias”. In: *Machine learning* 10.2 (1993), pp. 153–178.