Modéles formels pour l'IA explicable : des explications pour les arbres de décision

Louenas Bounia

UNIVERSITÉ D'ARTOIS

Modèles formels pour l'IA explicable : des explications pour les arbres de décision

THÈSE

présentée et soutenue publiquement le 22 décembre 2023

en vue de l'obtention du

Doctorat de l'Université d'Artois

(Spécialité Informatique)

par

Louenas BOUNIA

Encadrants :	Frédéric KORICHE	Université d'Artois - CRIL
	Pierre MARQUIS	Université d'Artois - CRIL
Rapporteurs :	Isabelle BLOCH	Sorbonne Université - LIP6
	Martin COOPER	Université Paul Sabatier - IRIT
Examinateur :	Céline ROUVEIROL	Université Sorbonne Paris Nord - LIPN

Centre de Recherche en Informatique de Lens – CNRS UMR 8188
Université d'Artois, rue Jean Souvraz, S.P. 18 F-62307, Lens Cedex France
Secrétariat : +33 (0)3 21 79 17 23
http://www.cril.univ-artois.fr

Mise en page avec memcril (B. Mazure, CRIL) et thloria (D. Roegel, LORIA).

Je dédie cette thèse à mon père.

Table des matières

Table des figures viii

Introduction générale 1

1 Introduction générale 1
- 1.1 La croissance du *machine learning* 1
- 1.2 Explicabilité 2
- 1.3 Explicabilité et interprétabilité 4
- 1.4 Types d'explications et méthodes d'explication 4
- 1.5 Contributions et questions de recherche 5

Partie I État de l'art

Chapitre 1 Logique propositionnelle et problème SAT 10

1.1 Notions de logique propositionnelle 10
- 1.1.1 Introduction à la logique propositionnelle classique 10
- 1.1.2 Les formules propositionnelles et les affectations 11
- 1.1.3 Fonctions booléennes et impliquants 13

1.2 Rappels de la théorie de la complexité 14
- 1.2.1 Principes de base et définitions 14
- 1.2.2 Rappels sur la machine de Turing 15
- 1.2.3 Classes de complexité et hiérarchie polynomiale 17
- 1.2.4 Classe de complexité #P 19

1.3 Le problème SAT 20

1.3.1 Formes normales . . . 20
1.3.2 Problème de satisfiabilité booléenne (SAT) . . . 21
1.3.3 Problème de satisfiabilité maximale (MaxSAT) . . . 21
1.3.4 Problème de comptage de modèles (# SAT) . . . 23
1.3.5 DNF orthogonale . . . 24
1.4 Conclusion . . . 25

Chapitre 2 Optimisation combinatoire

2.1 Introduction . . . 26
2.1.1 Définitions et propriétés clés . . . 27
2.1.2 Problèmes combinatoires NP-complets . . . 28
2.1.3 Matroïdes . . . 29
2.1.4 Optimisation linéaire . . . 31
2.2 Sous-modularité et super-modularité . . . 32
2.2.1 Fonctions sous-modulaires . . . 33
2.2.2 Fonctions super-modulaires . . . 35
2.3 Algorithmes approchés . . . 36
2.3.1 Généralités sur l'approximation . . . 37
2.3.2 Maximisation sous-modulaire . . . 38
2.3.3 Minimisation super-modulaire . . . 39
2.3.4 Importance de la courbure . . . 40
2.4 Conclusion . . . 42

Chapitre 3 Apprentissage automatique

3.1 Apprentissage automatique et classification . . . 43
3.1.1 Types d'apprentissage . . . 44
3.1.2 Apprentissage supervisé et classification . . . 44
3.1.3 Modèles d'apprentissage automatique . . . 46
3.2 Modèles symboliques (basés sur des règles) . . . 47
3.2.1 Arbres de décision (*decision trees*) . . . 48
3.2.2 Listes de décisions (*decision lists*) . . . 50
3.2.3 Règles de décisions (*decision rules*) . . . 50
3.3 Modèles ensemblistes . . . 52
3.3.1 Forêts aléatoires . . . 53
3.3.2 Boosting d'arbres . . . 54
3.3.3 Quelques statistiques sur les arbres de décision et les ensembles d'arbres . . . 55
3.4 Conclusion . . . 58

Chapitre 4 Explicabilité des prédictions en apprentissage automatique

4.1 Panorama des explications 60
4.1.1 Introduction 60
4.1.2 Formes d'explications 61
4.1.3 Variabilité des formes d'explications 62
4.1.4 Propriétés des explications 62
4.2 Méthodes agnostiques locales « orientées boîtes noires » 64
4.2.1 LIME 64
4.2.2 SHAP (*SHapley Additive exPlanations*) 66
4.2.3 Anchors 69
4.2.4 Limites et faiblesses des approches agnostiques de l'état de l'art 71
4.3 Méthodes formelles orientées modèles 71
4.3.1 Explications abductives 72
4.3.2 Explications contrastives 73
4.3.3 Notes complémentaires 73
4.4 Conclusion 74

Conclusion de la partie « état de l'art »

Partie II Contributions

Introduction **78**

Chapitre 5 Sur l'intelligibilité computationnelle des classifieurs booléens

5.1 Requêtes XAI 81
5.1.1 Introduction 81
5.1.2 Requêtes de vérification 82
5.1.3 Requêtes d'explication 83
5.1.4 Perceptrons multi-couches booléens « PMC » et réseaux de neurones binaires « BNN » 84
5.2 Complexité des requêtes XAI pour les classifieurs booléens 86
5.2.1 Le cas des requêtes de vérification 87

5.2.2 Le cas des requêtes d'explication . . . 89
5.3 Sur l'intelligibilité des classifieurs booléens . . . 90
5.3.1 Sur l'intelligibilité des requêtes XAI pour les arbres de décision . . . 91
5.3.2 Sur l'intelligibilité des requêtes XAI pour les classifieurs booléens . . . 94
5.4 Discussion . . . 99
5.5 Conclusion . . . 100

Chapitre 6 Le pouvoir explicatif des arbres de décision 101
6.1 Explications abductives et contrastives . . . 102
6.1.1 Transformation d'arbres de décision binaires en forme normale . . . 102
6.1.2 Explications abductives . . . 103
6.1.3 Explications contrastives . . . 104
6.2 Calculer toutes les explications abductives . . . 105
6.2.1 Énumérer toutes les raisons suffisantes minimales . . . 107
6.2.2 Synthétiser l'ensemble des raisons suffisantes . . . 111
6.3 Calculer toutes les explications contrastives . . . 115
6.4 Expérimentations . . . 118
6.5 Conclusion . . . 124

Chapitre 7 Explications abductives préférées 125
7.1 Explications abductives préférées . . . 126
7.1.1 Modèles de préférences . . . 126
7.1.2 Explications abductives préférées pour les arbres de décision . . . 128
7.2 L'impact des fonctions de poids sur les raisons abductives préférées . . . 129
7.2.1 Fonctions de poids . . . 130
7.2.2 Transformation monotone . . . 131
7.3 Résultats expérimentaux . . . 131
7.3.1 Protocole expérimental . . . 131
7.3.2 Résultats expérimentaux . . . 133
7.4 Conclusion . . . 135

Chapitre 8 Approximation des explications probabilistes par minimisation super-modulaire 136

8.1 Introduction . . . 137
8.2 Explication probabiliste . . . 138
8.2.1 Explication probabiliste . . . 138
8.2.2 Formulation du problème . . . 139
8.2.3 Évaluation des erreurs d'explication . . . 140

8.3 Minimisation super-modulaire . . . 141
8.3.1 Minimisation de l'erreur d'explication . . . 142
8.3.2 Minimisation de l'erreur d'explication non normalisée . . . 143
8.4 Algorithmes approchés . . . 144
8.4.1 Descente gloutonne (GD) . . . 144
8.4.2 Ascension gloutonne (GA) . . . 145
8.4.3 Application aux arbres de décision . . . 147
8.5 Expérimentations . . . 148
8.5.1 Protocole expérimental . . . 148
8.5.2 Résultats . . . 150
8.5.3 Expérimentations supplémentaires . . . 150
8.6 Discussion . . . 153
8.7 Conclusion . . . 154

Conclusion de la partie « contributions » **156**

Conclusion générale **158**

Bibliographie **161**

Table des figures

1 Deux objectifs principaux pour l'XAI : justification et validation (source : XAI today). . 3
2 Schéma résumant comment associer une boîte noire à une boîte blanche pour expliquer des prédictions. . . . 6

3 Une machine de Turing (source : turing) . . . 16
4 Classes de complexité NP, CoNP (source : [Mos17]) . . . 18
5 Diagramme de Hasse de la hiérarchie polynomiale (les flèches indiquent l'inclusion entre les classes de complexité). . . . 19

6 5-fold cross validation (source : [PVG+11]). . . . 46
7 Une structure générale d'arbre de décision (source : DT) . . . 48
8 Un arbre de décision T pour classer l'attribution d'un *prêt bancaire* . . . 49
9 Règles de décision. . . . 51
10 Une architecture des méthodes d'ensemble (source : ensemble) . . . 52
11 Une architecture des méthodes d'ensemble (source : bagging) . . . 53
12 Une forêt aléatoire F pour décider d'accorder ou non un prêt à un client. . . . 54
13 Un modèle d'arbres boostés B pour décider d'accorder ou non un prêt à un client. Les poids des arbres sont respectivement de $0,5$, $0,25$ et $0,25$. . . . 55
14 Scatter plot des tailles des raisons directes pour 4 benchmarks. . . . 57

15 LIME (source : [RSG16]) . . . 65
16 Exemple d'une explication d'une mauvaise classification d'un husky « Husky vs Wolf » (source : [RSG16]). . . . 66
17 Exemple pour illustrer SHAP (source : shap) . . . 67
18 Temps de calcul des valeurs de Shapley selon le nombre d'instances TreeSHAP vs KernelSHAP pour le *dataset* « compas ». . . . 69
19 LIME vs. Anchors. Une visualisation (source : anchor) . . . 70
20 Une représentation du classifieur h par un arbre de décision. . . . 72

21 Un exemple de perceptron multi-couches booléen . . . 85
22 L'arbre de décision φ. En rouge, les valeurs de $w(N)$ de chaque noeud. . . . 93
23 En bleu : les identifiants de chaque noeud. . . . 93

24 Un arbre de décision T pour reconnaître les orchidées Cattleya. . . . 103
25 Un arbre de décision T équivalent à $x_1 \vee x_2$. . . . 106

26 Une instance mnist49 (à gauche), puis (de gauche à droite) deux raisons suffisantes de cette instance, dont une de taille minimale (la plus à gauche des deux), et une carte de chaleur explicative pour l'instance (la figure la plus à droite). Les pixels formant l'instance « 4 » sont données en arrière plan mais ne font pas partie de la raison suffisante. 106
27 Un circuit d-DNNF représentant les raisons suffisantes pour $\boldsymbol{x}$ étant T. 115
28 Deux exemples provenant de mnist49 : un « 4 » correctement identifié comme un « 4 » par l'arbre de décision, et un « 9 » mal classifié comme un « 4 » par l'arbre de décision. Pour chaque instance, deux explications contrastives sont présentées (les premières à gauche correspondent au « 4 » correctement classifié, et les dernières à droite correspondent au « 9 » mal classifié). 117
29 Comparaison du nombre de raisons suffisantes avec le nombre de raisons suffisantes minimales pour 4 datasets. 121
30 Comparaison des tailles des raisons suffisantes avec les tailles des raisons suffisantes minimales pour les instances de 4 datasets. 122
31 Distributions des temps de calcul nécessaires pour compter les raisons suffisantes et les raisons suffisantes minimales pour 1000 instances provenant de deux datasets. 123

32 Un arbre de décision pour classer les *prêts bancaires* en utilisant les attributs suivants : x_1 : « n'a pas de CDI », x_2 : « a plus de 50 ans », x_3 : « revenus annuels inférieurs à 35K » et x_4 : « n'a pas remboursé un précédent prêt ». 129
33 Temps de calcul requis pour calculer toutes les raisons suffisantes minimales et les raisons suffisantes de coût minimal pour les fonctions de poids « SHAP », « f_importance » et « wordfreq » pour le *dataset* « *employee* ». 132

34 L'erreur $\epsilon_{h,\boldsymbol{x}}(S)$ (en bleu) et le nombre d'erreurs $\mu_{h,\boldsymbol{x}}(S)$ (en magenta) pour chaque $S \subseteq [3]$, en utilisant le classifieur h donné par (8) et l'instance $\boldsymbol{x} = (1, 1, 1)$. 139
35 Une représentation par arbre de décision de (8). 140
36 L'erreur $\epsilon_{h,\boldsymbol{x}'}(S)$ (en bleu) et le nombre d'erreurs $\mu_{h,\boldsymbol{x}'}(S)$ (en magenta) pour chaque $S \subseteq [3]$, en utilisant le classifieur h donné par (8) et l'instance $\boldsymbol{x}' = (1, 1, 0)$ $(h(\boldsymbol{x}') = 0)$. 141
37 L'arbre de décision $\mathcal{T}_2$ utilisé dans la preuve de la proposition 26. 142
38 Diagrammes à barres pour les erreurs de GA (jaune) et GD (bleu), en utilisant $k = 7 \pm 2$. 151

Introduction générale

1 Introduction générale

Notre quotidien est de plus en plus impacté par l'utilisation des algorithmes de l'intelligence artificielle (IA), notamment ceux de l'apprentissage automatique (*machine learning*). Ces algorithmes jouent un rôle crucial dans une multitude de domaines. Les modèles d'apprentissage automatique, qui font partie des méthodes d'intelligence artificielle, sont utilisés pour prendre des données en entrée et prédire une sortie correspondante. Contrairement aux programmes informatiques traditionnels, qui sont explicitement programmés par des humains pour suivre une séquence prédéfinie d'instructions, les modèles d'apprentissage automatique se programment en partie eux-mêmes en apprenant à identifier des règles pertinentes dans les données. Ces modèles cherchent à généraliser les motifs repérés pour les appliquer à de nouvelles situations. Grâce à la puissance de calcul des ordinateurs modernes, les modèles d'apprentissage automatique peuvent analyser d'énormes quantités de données en temps réduit. Si les programmes informatiques traditionnels suffisent à automatiser des tâches simples, l'apprentissage automatique se révèle particulièrement adapté pour automatiser des tâches cognitives complexes, comme la traduction de texte et la reconnaissance d'objets dans des images.

Dans ces applications, l'apprentissage automatique prend des décisions de manière autonome. Pour des tâches à enjeu élevé et important, l'apprentissage automatique peut considérablement améliorer l'efficacité et la précision des prises de décision humaines. Toutefois, dans de nombreux cadres applicatifs, les modèles d'apprentissage automatique ont pour seule vocation d'aider l'utilisateur humain qui reste responsable de la décision finale. Le modèle d'apprentissage automatique peut ainsi agir comme un *super assistant* qui aide à la prise de décision en fournissant des informations pertinentes pour celle-ci.

Les modèles d'apprentissage automatique, notamment les réseaux de neurones profonds, doivent souvent être très complexes pour offrir de bonnes performances prédictives. Cependant, cette obsession pour la précision prédictive est critiquée, car elle oblige à construire des modèles volumineux et opaques. Le comportement de *boîte noire* de ces modèles empêche les utilisateurs de vérifier si les règles apprises correspondent à leurs attentes et intentions, ce qui soulève des questions éthiques et génère un manque de confiance. Le manque de transparence limite la diffusion des méthodes de l'apprentissage automatique, surtout dans des domaines critiques où comprendre le *pourquoi* d'une décision est essentiel. Ainsi, il existe une demande croissante d'approches permettant d'aider les utilisateurs à comprendre les systèmes d'IA et à saisir le raisonnement qui conduit aux prédictions réalisées.

1.1 La croissance du *machine learning*

Au cours des dernières décennies, l'apprentissage automatique a connu une croissance exponentielle qui a radicalement transformé le domaine de l'intelligence artificielle. Cette expansion fulgurante a été rendue possible par plusieurs facteurs clés. Tout d'abord, les avancées technologiques ont offert une

puissance de calcul accrue et la disponibilité de ressources informatiques massives a permis de traiter des volumes de données importants. Ensuite, l'explosion du **Big Data** a fourni un accès à de larges ensembles de données provenant de diverses sources, nourrissant ainsi les modèles d'apprentissage automatique avec des informations variées et riches. Enfin, dans l'histoire de l'informatique, plusieurs événements historiques ont joué un rôle crucial dans cette remarquable progression du *machine learning*. Dans les années 1950. **Alan Turing** a jeté les bases de l'intelligence artificielle [Tur50], avec ses travaux pionniers sur la calculabilité et les réseaux neuronaux. Une étape clé est survenue dans les années 2010 avec la percée du *deep learning* et des réseaux de neurones profonds. Ces architectures sophistiquées ont permis aux modèles d'apprentissage automatique de réaliser des performances spectaculaires dans des domaines tels que la vision par ordinateur et le traitement du langage naturel. L'utilisation de grandes quantités de données et la puissance de calcul des ordinateurs ont propulsé le *deep learning* vers de nouveaux sommets de précision et d'efficacité, ouvrant ainsi la voie à de nouvelles possibilités d'innovation et d'applications dans divers domaines.

Aujourd'hui, l'apprentissage automatique est devenu omniprésent dans notre vie quotidienne. Des applications courantes de l'apprentissage automatique incluent les assistants vocaux tels que *Siri* et *Alexa*, les recommandations de produits sur les sites de commerce électronique, les systèmes de recommandation sur les plateformes de streaming, et même les véhicules autonomes. L'efficacité et la commodité apportées par ces applications ont séduit les utilisateurs du monde entier. Cependant, cette croissance rapide n'est pas sans poser de défis. L'utilisation croissante des modèles d'apprentissage automatique dans des domaines critiques, tels que les soins de santé, la finance et la sécurité, a soulevé des questions sur l'éthique, la responsabilité et la transparence. En raison de leur complexité et de leur comportement de *boîte noire*, ces modèles peuvent être difficiles à comprendre et à expliquer. Cette opacité rend le besoin urgent de méthodes pour expliquer les décisions prises par les modèles. Ainsi, la demande croissante d'explications dans le domaine de l'apprentissage automatique, en particulier dans les domaines critiques, vise à renforcer la confiance des utilisateurs et à garantir une adoption responsable de cette technologie. La recherche se concentre désormais sur le développement de méthodes d'interprétabilité et d'explicabilité afin de mieux comprendre les mécanismes de prise de décision des modèles d'apprentissage automatique, ouvrant ainsi la voie à de nouvelles opportunités d'innovation et de progrès dans le domaine. En permettant aux utilisateurs de comprendre et d'interagir de manière transparente avec ces systèmes, nous pourrons exploiter pleinement le potentiel de l'apprentissage automatique pour créer un impact positif et durable dans divers secteurs.

1.2 Explicabilité

Le besoin et la motivation pour l'intelligence artificielle explicable (XAI) peuvent être résumés en deux objectifs principaux, comme illustré a la figure 1. Les explications permettent de justifier les résultats d'un modèle d'apprentissage automatique en fournissant un motif du raisonnement suivi par le modèle. Une fois que les explications sont extraites, il est essentiel de vérifier la validité de ces explications pour s'assurer qu'elles correspondent aux intentions et aux attentes. Les explications qui révèlent des schémas différents de ceux qui étaient attendus par l'utilisateur peuvent également conduire à la découverte de nouvelles connaissances. Les paragraphes suivants approfondissent ces objectifs.

Justification. La justification des prédictions est un aspect essentiel dans le domaine de l'apprentissage automatique [Coe19]. Il s'agit de répondre à la question « pourquoi le modèle d'apprentissage automatique a-t-il donné cette prédiction ? ». Comprendre le raisonnement derrière une décision devient particulièrement crucial lorsque le modèle aboutit à une prédiction erronée ou inattendue, ou encore dans des cas d'utilisation à forts enjeux où les prédictions peuvent avoir des conséquences considérables. Par

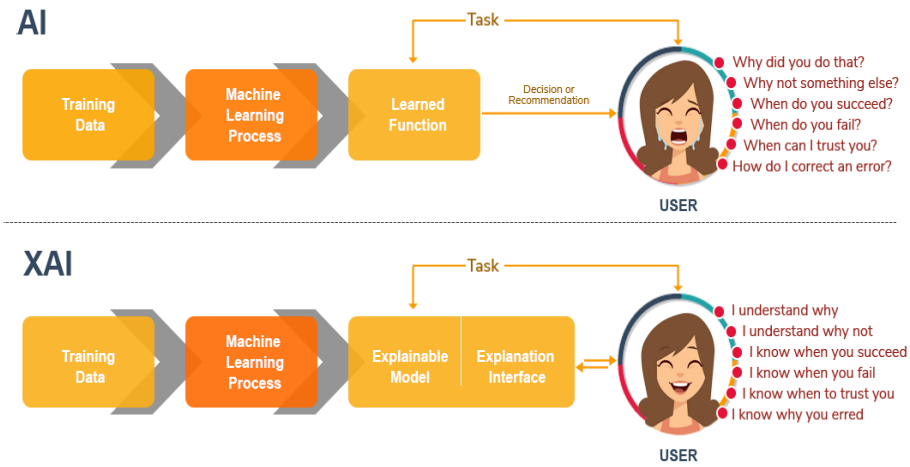


FIGURE 1 – Deux objectifs principaux pour l'XAI : justification et validation (source : XAI today).

exemple, il peut être primordial de comprendre pourquoi un traitement médical particulier est recommandé pour un patient, ou pourquoi une voiture a automatiquement freiné dans une situation donnée. En outre, la justification des décisions automatisées peut être une exigence légale. Le règlement général sur la protection des données (RGPD) de l'Union Européenne impose la nécessité de justifier les décisions automatisées, donnant ainsi aux personnes concernées pour une prise de décision d'obtenir des informations significatives sur la logique utilisée dans la prise de décision automatisée. Dans certains domaines spécifiques tels que l'évaluation du crédit bancaire, des réglementations plus concrètes exigent des raisons pour les refus de décisions basées sur des critères de crédit. En plus de son importance pour les personnes impactées pour la décision prise, l'explicabilité est également cruciale pour les utilisateurs qui interagissent avec les modèles d'apprentissage automatique. Le manque d'explicabilité peut être considéré comme un problème d'un point de vue moral, car il peut entraîner une ignorance chez les utilisateurs du modèle, entraînant une sur-dépendance à l'IA et le transfert de responsabilité. Les auteurs de [Coe19] soulignent que l'explicabilité est essentielle pour que les agents humains qui utilisent des systèmes d'IA puissent comprendre leurs actions, agir de manière responsable et assumer la responsabilité de leurs décisions. Ainsi, l'explicabilité joue un rôle fondamental dans l'évolution des interactions entre les humains et les systèmes d'IA.

Validation. L'IA explicable (XAI) doit permettre de vérifier si le modèle d'apprentissage automatique considéré a appris une stratégie de raisonnement adéquate. Elle doit pouvoir mettre en évidence les raccourcis, biais et autres problèmes présents dans les données d'entraînement. Le phénomène du « Clever Hans » [1] peut entraîner des performances trop optimistes en laboratoire et des comportements non intuitifs dans la pratique réelle. L'explicabilité permet aux utilisateurs d'évaluer la fiabilité des modèles. Le besoin de validation des modèles d'apprentissage automatique est crucial pour vérifier si le raisonnement du modèle répond aux intentions et attentes. Étant dépourvu de sens moral, l'apprentissage automatique nécessite des explications pour permettre aux humains de vérifier la bienveillance du modèle. Cette exigence d'explicabilité est particulièrement pertinente pour éviter des comportements non éthiques, surtout dans des domaines critiques tels que la santé, la finance et la sécurité. Les modèles d'apprentissage automatique peuvent apprendre des biais indésirables présents dans les données d'entraînement, ce qui peut

1. Le phénomène du « Clever Hans » [SG13] désigne une illusion cognitive où un animal ou un individu semble résoudre un problème complexe mais répond en réalité à des indices subtils du contexte.

nuire aux groupes sous-représentés. L'explicabilité sert ainsi de mesure de protection pour vérifier si le raisonnement du modèle est conforme aux normes éthiques et réglementaires. En rendant explicite le raisonnement derrière les décisions, l'explicabilité favorise aussi les débats éthiques.

1.3 Explicabilité et interprétabilité

Les termes d'IA explicable (XAI) et d'apprentissage automatique interprétable, ainsi que l'explicabilité et l'interprétabilité (et dans une certaine mesure l'intelligibilité), sont étroitement liés, bien qu'il n'y ait pas de consensus sur la distinction exacte entre ces concepts. On peut considérer qu'une méthode d'explication « explique » en transmettant une explication, tandis qu'un humain « interprète » en recevant cette explication, ce qui indique que le bon verbe dépend du sujet principal. Par conséquent, une explication peut être considérée comme le produit central d'un système explicatif, ou comme faisant partie du processus social et cognitif de l'utilisateur. Ainsi, le terme « intelligence artificielle explicable » est principalement utilisé pour désigner l'ensemble des méthodes impliquant une étape explicite d'explication nécessaire à la compréhension d'un modèle prédictif d'apprentissage automatique. Par contraste, « l'apprentissage automatique interprétable » est davantage associé aux modèles d'apprentissage automatique qui ne nécessitent pas d'étape explicite distincte pour expliquer, mais qui effectuent des prédictions en utilisant un processus de raisonnement intrinsèquement compréhensible.

Fournir une explication consiste à présenter les différents aspects du raisonnement, du fonctionnement et/ou du comportement d'un modèle d'apprentissage automatique de manière compréhensible pour un humain. Les termes « raisonnement, fonctionnement et/ou comportement » englobent différents types d'explications et de méthodes d'explication. Le raisonnement se réfère au processus par lequel le modèle prend une décision spécifique, répondant à des questions telles que « pourquoi le modèle a-t-il produit cette sortie particulière pour cette entrée spécifique ? ». Ces explications locales ciblent une instance de données spécifique et justifient une seule prédiction. Le fonctionnement concerne les mécanismes internes et les structures de données internes des modèles d'apprentissage automatique. Le comportement se réfère à la manière dont le modèle fonctionne globalement sans nécessiter une analyse spécifique des mécanismes internes. Alors qu'une explication locale est générée pour un échantillon de données particulier, une explication globale cible l'ensemble du modèle prédictif et vise à prendre compte de n'importe quelle entrée.

Explications locales et globales. Il est nécessaire d'établir une distinction en fonction des prédictions pour lesquelles les explications sont générées. D'un côté, les approches d'interprétabilité globale visent à fournir des explications pour aider l'utilisateur à acquérir une connaissance globale du modèle. Les informations extraites concernent donc le comportement général du modèle. Par exemple, les coefficients d'une régression linéaire fournissent des informations sur l'impact de chaque caractéristique à l'échelle globale. D'un autre côté, les approches locales visent à générer des explications pour une prédiction spécifique liée à une instance particulière en entrée.

1.4 Types d'explications et méthodes d'explication

Ces dernières années ont été marquées par une multiplication des méthodes d'explication. Ces méthodes génèrent des explications de différentes manières, aboutissant à divers types d'explications adaptées à différents types de données et de modèles d'apprentissage automatique. Nous pouvons identifier plusieurs types d'explications, chacun présentant un format spécifique. Outre le format des explications, un autre aspect crucial est la méthode de production de ces explications. En général, nous pouvons distin-

guer deux approches différentes pour expliquer un modèle d'apprentissage automatique, qui permettent de générer des explications locales ou globales.

Méthodes d'explication post-hoc. Les méthodes d'explication post-hoc fournissent des explications pour un modèle d'apprentissage automatique déjà entraîné. Elles peuvent donner une explication globale du raisonnement du modèle ou générer des explications locales pour expliquer une instance spécifique. Par exemple, les cartes de chaleur peuvent mettre en évidence les parties importantes d'une image pour la prédiction du modèle, les scores d'importance peuvent indiquer les mots pertinents pour la classification d'un texte, et des règles de décision sous forme de clauses ou de termes logiques peuvent également être utilisées pour expliquer. Une explication globale post-hoc pourrait prendre la forme d'un ensemble de règles qui reproduit le raisonnement du modèle et agit comme un substitut pour la boîte noire.

Les approches auto-explicatives « *self-explaining models* ». Les méthodes d'interprétabilité intégrées, également appelées méthodes d'explication ante-hoc, intègrent l'interprétabilité directement dans le modèle d'apprentissage. Cette interprétabilité intégrée peut être soit partielle au niveau local, par exemple, en utilisant des mécanismes d'attention pour indiquer quels attributs d'entrée sont pertinents pour les prédictions du modèle, soit globale. Un exemple bien connu de modèle interprétable est le modèle de régression linéaire : les coefficients associés à chaque attribut indiquent son influence. Cependant, un inconvénient potentiel de l'interprétabilité intégrée est que son incorporation dans le modèle peut limiter son architecture et potentiellement influencer ses performances.

1.5 Contributions et questions de recherche

Sujet de thèse et questions de recherche. Nos recherches visent à expliquer des systèmes d'IA (des modèles d'apprentissage automatique) considérés et vus comme des systèmes complexes. Dans cette étude, nous explorons l'approche consistant à associer à un modèle d'apprentissage automatique considéré comme une boîte noire, une boîte blanche (ou une boîte transparente), sous la forme d'un circuit (ou une formule) ayant le même comportement que le modèle d'apprentissage en termes d'entrées/sorties. L'intérêt de cette *boîte blanche* est de constitue une base pour répondre aux demandes d'explication et de vérification de l'XAI, comme illustré dans le schéma de la figure 2. Ces requêtes sont posées pour estimer la robustesse des décisions prises ou des prédictions effectuées. Dans cette thèse, plusieurs questions ont été abordées. En particulier :

— Quel type de boîte blanche faut-il considérer et pour quel type de boîte noire ?
— Quels encodages sont-ils possibles pour passer de la boîte noire considérée à une boîte blanche ?
— Quelle information utile faut-il préserver pour pouvoir réaliser ensuite des requêtes d'explication ou de vérification ?
— Quel est le coût calculatoire de ces requêtes selon le type de circuit retenu et la représentation choisie pour ce circuit ?
— Comment prendre en considération les limitations cognitives et les connaissances des utilisateurs ?

La boîte blanche étant indépendante des entrées de la boîte noire, elle peut être pré-traitée (compilée) afin de faciliter la génération d'explications des prédictions réalisées par la boîte noire et l'évaluation de sa fiabilité. Les questions de recherche énumérées précédemment sont étudiées dans cette thèse, qui se concentre sur les aspects de modélisation, tels que les liens entre les boîtes noires et les boîtes blanches, leurs encodages et leurs propriétés.

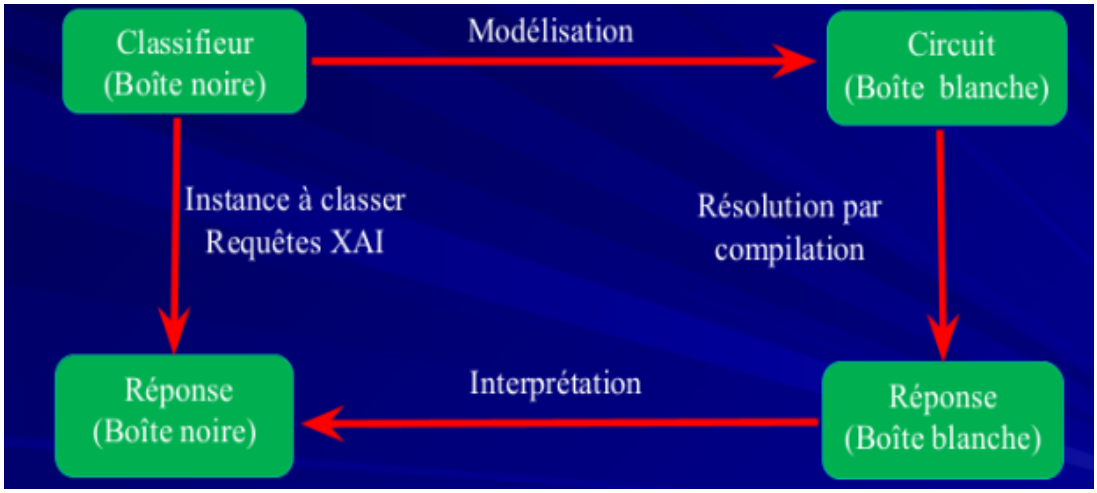


FIGURE 2 – Schéma résumant comment associer une boîte noire à une boîte blanche pour expliquer des prédictions.

Contributions. Au cours de cette thèse, nous avons travaillé essentiellement sur des méthodes de génération d'explications locales post-hoc. Nous nous sommes concentrés sur les structures d'apprentissage automatique qui sont basées sur les arbres, à savoir les arbres de décision (ce sont aussi des formes compilées particulières). Nous avons entamé nos travaux en étudiant l'intelligibilité computationnelle des classifieurs booléens (chapitre 5), caractérisés par leur capacité à répondre aux requêtes XAI en temps polynomial. Nous avons examiné un grand nombre de classifieurs, allant des arbres de décision aux réseaux de neurones en passant par les classifieurs à base d'ensemble, et nous avons constaté qu'il existe un écart significatif d'intelligibilité entre ces familles de classifieurs. En effet, toutes les requêtes XAI que nous avons considérées peuvent être traitées en temps polynomial pour les arbres de décision, tandis que pour le reste des familles de classifieurs, aucune d'entre elles n'est traitable en temps polynomial dans le pire des cas (sauf si P=NP).

Ensuite, nous avons développé une approche pour expliquer les arbres de décision (chapitre 6), en nous inspirant des travaux antérieurs démontrant l'existence d'explications plus courtes que le chemin correspondant à l'instance considérée dans l'arbre. Grâce à un algorithme glouton, nous avons réussi à calculer de manière plus simple une raison suffisante pour la décision prise, qui est une explication abductive irredondante. Nous avons également découvert comment énumérer ces raisons suffisantes et introduit les concepts d'attributs nécessaires / pertinents et de pouvoir explicatif d'un littéral. Nous nous sommes ensuite penchés sur la question de la recherche d'explications de taille minimale, en estimant qu'une explication concise avec moins d'arguments est souvent plus compréhensible pour un opérateur humain. Cependant, le nombre d'explications, même celles de taille minimale peut être élevé. C'est pourquoi nous avons réfléchi à l'amélioration de nos algorithmes de calcul d'explications en prenant en compte des préférences (chapitre 7), qu'elles soient renseignées manuellement par un utilisateur ou issues d'un pré-traitement. Malgré tout, en raison des limitations cognitives des utilisateurs, les explications abductives préférées peuvent être de trop grande taille pour être facilement interprétables. Pour cette raison, nous avons développé une approche pour réduire la taille des explications abductives produites tout en déterminant l'étiquette prédite avec une probabilité élevée (chapitre 8). À cet effet, nous avons proposé une méthode pour calculer des explications probabilistes en utilisant une approche super-modulaire.

Collaboration. Pendant la réalisation de ce travail de recherche et dans le cadre du projet **EXPEKCTATION** http://www.cril.univ-artois.fr/expekctation/, j'ai eu le privilège de collaborer étroitement avec **Steve Bellart**, un collègue qui a beaucoup apporté à nos résultats communs. Sa profonde compréhension de ces domaines ainsi que son expertise technique ont grandement contribué à la progression du projet. Ensemble, nous avons travaillé sur les techniques d'explication des décisions prises par les modèles d'apprentissage automatique. La collaboration avec Steve sur ces modèles et l'explicabilité des décisions prises a été un véritable atout pour le bon déroulement du projet, notre compréhension du sujet et a conduit à des résultats significatifs.

Organisation du document. La suite de ce mémoire est organisée en deux parties. la première partie « état de l'art » est composée de quatre chapitres, visant à fournir un ensemble de préliminaires formels (définitions, notations, résultats, etc.) utilisés dans les chapitres de la seconde partie du mémoire et de positionner nos résultats dans le contexte de l'apprentissage automatique et de l'IA explicable. La seconde partie « contributions » est elle aussi formée de quatre chapitres, dans lesquels les apports de cette thèse sont présentés.

Première partie

État de l’art

1

Logique propositionnelle et problème SAT

Sommaire

1.1 Notions de logique propositionnelle 10
1.1.1 Introduction à la logique propositionnelle classique 10
1.1.2 Les formules propositionnelles et les affectations 11
1.1.3 Fonctions booléennes et impliquants 13
1.2 Rappels de la théorie de la complexité 14
1.2.1 Principes de base et définitions 14
1.2.2 Rappels sur la machine de Turing 15
1.2.3 Classes de complexité et hiérarchie polynomiale 17
1.2.4 Classe de complexité #P 19
1.3 Le problème SAT 20
1.3.1 Formes normales 20
1.3.2 Problème de satisfiabilité booléenne (SAT) 21
1.3.3 Problème de satisfiabilité maximale (MaxSAT) 21
1.3.4 Problème de comptage de modèles (# SAT) 23
1.3.5 DNF orthogonale 24
1.4 Conclusion 25

Dans ce chapitre, nous présentons d'abord les notations et concepts de la logique propositionnelle, ainsi que ceux de la théorie de la complexité qui sont utilisés tout au long de ce mémoire.

Nous nous penchons ensuite sur les problèmes de satisfaisabilité d'une formule propositionnelle (SAT), de satisfaisabilité maximale (MAX-SAT) et de comptage de modèles (#SAT). Nous décrivons également différentes formes normales, notamment les formes normales conjonctives (CNF) et disjonctives (DNF). Nous accordons enfin une attention particulière à la famille des DNF orthogonales.

1.1 Notions de logique propositionnelle

Une partie de nos résultats s'appuyant sur des connaissances représentées en logique propositionnelle, nous présentons d'abord les notions de formule et d'affectation.

1.1.1 Introduction à la logique propositionnelle classique

La logique propositionnelle classique est une branche de la logique qui se concentre sur l'étude des propositions, c'est-à-dire des énoncés qui peuvent être considérés comme vrais ou faux. Elle fournit un

cadre formel pour analyser et manipuler ces propositions de manière cohérente. La logique propositionnelle remonte à l'Antiquité, mais son développement formel et son utilisation systématique ont pris de l'ampleur au cours des siècles.

Il a fallu attendre les travaux de **George Boole** au XIXe siècle pour introduire des bases formelles et symboliques à la logique propositionnelle. **George Boole** a créé un système algébrique dans lequel les propositions peuvent être représentées par des variables et manipulées à l'aide d'opérations logiques. Son livre ***An Investigation of the Laws of Thought (1854)*** est considéré comme une étape importante dans le développement de la logique mathématique. Au XXe siècle, la logique propositionnelle a été formalisée de manière rigoureuse dans le cadre de la logique mathématique et est devenue une composante essentielle des fondements des mathématiques et de l'informatique.

La logique propositionnelle joue un rôle essentiel dans plusieurs cadres pour diverses raisons. Dans le domaine de l'informatique, elle est utilisée pour la conception de circuits, la programmation et la vérification des logiciels. Elle est également cruciale en intelligence artificielle et en apprentissage automatique, permettant la modélisation des connaissances et la résolution de problèmes complexes. En mathématiques, elle est employée pour la démonstration de théorèmes.

La représentation des connaissances en logique propositionnelle. Dans le contexte de ce mémoire, nous serons conduits à calculer des explications. Il existe différents types d'explications, et ces explications expriment des connaissances sous différentes formes. Une grande partie de ces explications peut être exprimée sous forme textuelle, d'images, etc. Cependant, le langage que nous utilisons pour représenter des explications est celui de la logique propositionnelle, le choix est motivé par plusieurs raisons. Tout d'abord, le langage de la logique propositionnelle permet une représentation formelle des connaissances. De plus, la logique propositionnelle est relativement simple, facilitant ainsi l'analyse et la manipulation des connaissances. Les règles de déduction claires permettent des raisonnements logiques rigoureux. Enfin, son langage formel précis permet d'éviter les ambiguïtés et facilite la communication et la transmission des connaissances. En somme, choisir la logique propositionnelle pour présenter des connaissances permet de développer une approche formelle, claire et automatisable, favorisant des raisonnements logiques rigoureux.

Exemple 1 (**Représentation de connaissance en logique**). *Supposons que nous voulions représenter la connaissance « Si une personne est un étudiant et elle réussit tous ses examens, alors elle obtiendra son diplôme ». Nous pouvons utiliser les propositions atomiques suivantes :*

- P *: une personne est un étudiant*
- Q *: une personne réussit tous ses examens*
- R *: une personne obtient son diplôme*

nous pouvons exprimer cette connaissance comme suit : (P *et* Q*) implique* R*. Cette représentation indique que si les propositions* P *(être un étudiant) et* Q *(réussir tous les examens) sont toutes deux vraies, alors la proposition* R *(obtenir son diplôme) sera vraie.*

1.1.2 Les formules propositionnelles et les affectations

Formules. L'alphabet de la logique booléenne est constitué de symboles pour représenter différentes opérations. Les constantes propositionnelles « faux » et « vrai » sont représentés par $\bot$ et $\top$ respectivement. Les connecteurs de conjonction, de disjonction et de négation sont représentés par $\wedge$, $\vee$ et $\neg$ respectivement. Tandis que les connecteur d'implication, de bi-implication, ou-exclusif et sont représentés par $\Rightarrow$, $\Leftrightarrow$, $\oplus$ respectivement. Dans le contexte de la logique propositionnelle, le langage est l'ensemble des expressions bien formées qui peuvent être construites à partir des connecteurs et d'un ensemble fini V de variables booléennes. Le langage de la logique propositionnelle consiste en toutes les combinaisons

valides de variables et d'opérations logiques qui peuvent être utilisées pour exprimer des propositions logiques.

Exemple 2 (**formule propositionnelle**). *La formule :*

$$\phi_1 = (((x_1 \wedge x_2) \vee (x_1 \oplus x_3)) \Rightarrow x_4) \Leftrightarrow (\neg x_3 \Rightarrow (x_1 \oplus x_4))$$

est une formule de la logique propositionnelle sur l'ensemble de variables $V_1 = \{x_1, x_2, x_3, x_4\}$ *mais également sur tout sur-ensemble de cet ensemble* V_1.

Dans tout le mémoire, nous considérons les sous-ensembles de variables de l'ensemble dénombrable $\mathbf{V} = \{x_i : i \in \mathbb{N}\}$. Les variables sont ordonnées par la relation $i < j \Leftrightarrow x_i < x_j$. L'ensemble des variables booléennes apparaissant dans une formule ϕ de la logique propositionnelle est noté $Var(\phi)$. Pour la formule ϕ_1, nous avons $V_1 = Var(\phi_1)$. Pour un ensemble E quelconque de variables, nous notons $|E|$ le nombre d'éléments dans E, et nous avons donc $|V_1| = 4$.

Définition 1 (**littéral**). *Un littéral (noté* l*) est une variable booléenne ou sa négation.* $x \in \mathbf{V}$ *est un littéral positif, tandis que* $\neg x$ *est un littéral négatif.* x *et* $\neg x$ *sont des littéraux complémentaires.*

Nous rappelons qu'un littéral est dit monotone (ou pur) pour une formule booléenne ϕ si et seulement si l apparaît dans ϕ et que son complémentaire n'apparaît pas. Nous notons $Lit(\phi)$ l'ensemble des littéraux qui apparaissent dans ϕ. Par exemple, pour ϕ_1, nous avons $Lit(\phi_1) = \{x_1, x_2, x_3, \neg x_3, x_4\}$. Nous pouvons dire que le littéral $l = x_1$ est monotone pour la formule ϕ_1, tandis que $l = x_3$ ne l'est pas. Nous précisons que le principe de monotonie n'a de sens véritable que pour les formules en `NNF` (que nous allons définir).

Affectations, affectations partielles et termes. Une affectation z à un ensemble fini de variables $V \subseteq \mathbf{V}$ est une application qui associe chaque variable x de V à une valeur binaire, soit 0 soit 1. Ainsi, pour toute variable $x \in V$, nous notons $z[x]$ la valeur affectée par l'affectation z à la variable x. Soit maintenant z une affectation de V et V' un sur-ensemble de V. Nous définissons un prolongement de z sur V' comme étant toute affectation z' à V' qui satisfait $\forall x \in V, z[x] = z'[x]$. Une instance est une affectation z de toutes les variables de V, autrement dit, une instance est un vecteur $z \in \{0,1\}^{|V|}$. Enfin, tout au long de ce mémoire, nous considérons la notion d'affectation partielle à un ensemble de variables de V. Pour cela, nous introduisons le symbole * qui a la signification intuitive de « indéfini » (représentant soit 0, soit 1). Nous définissons une affectation partielle à V comme une fonction de V dans $\{0, 1, *\}$. Nous adoptons les mêmes notations que pour les affectations pour définir une instance partielle. Une instance partielle est ainsi représentée par un vecteur z' appartenant à $\{0, 1, *\}^{|V|}$, où $z'_i = *$ indique que la i-ème caractéristique de z' est indéfinie. Une affectation complète est donc une instance. Un terme est une conjonction de littéraux. Nous considérons le terme correspondant à une affectation $z \in \{0,1\}^d$ comme le terme défini comme suit : $t_z = \bigwedge_{i=1}^{d} x_i^{z_i}$ où $x_i^0 = x_i$ et $x_i^1 = \neg x_i$. Un terme t couvre une affectation z si $t \subseteq t_z$.

Exemple 3 (**affectation**). *Soit* V *l'ensemble de variables défini dans l'exemple 2, l'application* m_1 *définie par :* $m_1[x_1] = 0, m_1[x_2] = 0, m_1[x_3] = 1, m_1[x_4] = 1$ *est une affectation à l'ensemble* V*, tandis que l'affectation* z'_1 *définie par :* $z'_1[x_1] = 0, z'_1[x_2] = *, z'_1[x_3] = *, z'_1[x_4] = 1$*, est une affectation partielle à* V*. Elle est également représentée par* $(0, *, 1, *)$.

Modèles. Une affectation z à l'ensemble des variables booléennes $Var(\phi)$ est un modèle (noté m) de la formule propositionnelle ϕ si, lorsque nous évaluons ϕ en utilisant l'interprétation fournie par z, ϕ est évaluée à vrai (1). On note $z \models \phi$. Maintenant, pour une formule propositionnelle ϕ, nous notons $\mathcal{M}(\phi) \subseteq \{0,1\}^{Var(\phi)}$ l'ensemble de tous les modèles de ϕ.

Exemple 4 (modèles et terme). *Soit ϕ_1 la formule propositionnelle définie dans l'exemple 2. L'affectation $z = (1, 1, 1, 1)$ est un modèle de ϕ_1 car lorsque nous évaluons ϕ_1 en utilisant l'interprétation fournie par z, la formule est évaluée à vrai. On peut également écrire $\phi_1(z) = 1$ ou bien $z \models \phi_1$. Le terme correspondant à l'affectation z est défini comme suit : $t_z = x_1 \wedge x_2 \wedge x_3 \wedge x_4$ (ou encore $t_z = \{x_1, x_2, x_3, x_4\}$.*

- une formule ϕ est dite satisfiable (ou consistante) si elle admet au moins un modèle, et contradictoire sinon
- si $\phi = \neg\phi_1$ alors $m \models \phi \Leftrightarrow m_1 \not\models \phi_1$
- si $\phi = \phi_1 \wedge \phi_2 \cdots \wedge \phi_d$ alors $m \models \phi \Leftrightarrow \forall i \in \{1, \ldots, d\}\ m \models \phi_i$
- si $\phi = \phi_1 \vee \phi_2 \cdots \vee \phi_d$ alors $m \models \phi \Leftrightarrow \exists i \in \{1, \ldots, d\}\ m \models \phi_i$
- si $\phi = \phi_1 \oplus \phi_2 \cdots \oplus \phi_d$ alors $m \models \phi \Leftrightarrow |\{i \in \{1, \ldots, d\} : m \models \phi_i\}|$ est impair
- ϕ_2 est une conséquence logique de ϕ_1, noté $\phi_1 \models \phi_2$ ssi tout modèle de ϕ_1 est un modèle de ϕ_2
- deux formules ϕ_1 et ϕ_2 sont équivalentes (noté $\phi_1 \equiv \phi_2$) si elles ont les mêmes modèles.
- soit deux formules propositionnelles ϕ et ϕ' définies sur le même ensemble de variables, on a : $\mathcal{M}(\phi \wedge \phi') = \mathcal{M}(\phi) \cap \mathcal{M}(\phi'), \mathcal{M}(\phi \vee \phi') = \mathcal{M}(\phi) \cup \mathcal{M}(\phi'), \mathcal{M}(\phi \oplus \phi') = \mathcal{M}(\phi) \triangle \mathcal{M}(\phi')$.

1.1.3 Fonctions booléennes et impliquants

Fonctions booléennes. Les fonctions booléennes sont utilisées non seulement dans la modélisation des circuits logiques, mais également en théorie de la complexité, en théorie de l'information, et plus particulièrement en codage et en cryptographie. En théorie de l'information, les fonctions booléennes sont utilisées pour représenter et manipuler des informations binaires.

Définition 2 (fonction booléenne). *Soit $d \in \mathbb{N}^*$. Toute application $f : \{0,1\}^d \longrightarrow \{0,1\}$ est appelée fonction booléenne.*

Pour un entier d, soit $[d]$ l'ensemble $\{1, \ldots, d\}$. $\mathcal{F}_d$ désigne l'ensemble de toutes les fonctions booléennes de $\{0,1\}^d$ vers $\{0,1\}$, et nous utilisons dans la suite de ce mémoire $X_d = \{x_1, \ldots, x_d\}$ pour représenter l'ensemble des variables booléennes d'entrée pour la fonction booléenne f. $\mathbb{B}$ désigne $\{0,1\}$ et $\mathbb{B}^d$ désigne l'espace $\{0,1\}^d$. Une instance $\boldsymbol{x} \in \{0,1\}^d$ est appelée une instance positive si $f(\boldsymbol{x}) = 1$ et négative lorsque $f(\boldsymbol{x}) = 0$.

Impliquants et impliquants premiers. Une notion syntaxique importante, voire très importante, dans notre travail et dans ce mémoire, est la notion d'**impliquant premier**. Pour une fonction booléenne f, on dit qu'un terme t est un impliquant de f si $t \models f$.

Définition 3 (impliquant premier). *Soit $f \in \mathcal{F}_d$ une fonction booléenne et t un terme. t est un impliquant premier de f si et seulement si $t \models f$ et $\forall l \in t, t \setminus \{l\} \not\models f$. En d'autres termes, un impliquant premier est un terme qui exprime des conditions suffisantes pour satisfaire une fonction booléenne, sans qu'aucune de ses parties ne vérifie la même propriété (t est donc irredondant).*

Exemple 5 (impliquant premier). *Soit $f(x_1, x_2, x_3) = (x_1 \wedge x_2 \wedge x_3) \vee \neg x_3$ une fonction booléenne dans $\mathcal{F}_3$. L'affectation $z = (1, 1, 1)$ est un modèle de f ($f(z) = 1$), et le terme $t = x_1 \wedge x_2 \wedge x_3$ correspondant à cette affectation est un impliquant de f. Cependant, t n'est pas un impliquant premier car le terme $x_1 \wedge x_2$ qui est un sous-terme de t est lui-même un impliquant de f. En fait, $x_1 \wedge x_2$ est un impliquant premier de f.*

Il est à noter que le nombre d'impliquants d'une fonction booléenne peut être exponentiel par rapport au nombre de variables d'entrée [CH11]. Pour une fonction booléenne avec d variables d'entrée, le

nombre total d'impliquants peut être exponentiel en d. Cela est dû au fait qu'il existe 3^d combinaisons possibles des valeurs des variables d'entrée, et chaque combinaison peut potentiellement être représentée par un impliquant distinct. Ainsi, à mesure que le nombre de variables d'entrée augmente, le nombre de combinaisons et, par conséquent, le nombre d'impliquants peut augmenter de manière exponentielle.

1.2 Rappels de la théorie de la complexité

Dans ce mémoire, nous examinons la plupart des problèmes sous l'angle de l'optimisation et de l'algorithmique. Par conséquent, il est naturel que nous nous intéressions à la complexité de ces problèmes. Cette section a pour objectif de rappeler les notions fondamentales nécessaires pour situer correctement la complexité des problèmes que nous étudions. Nous ne rappelons pas ici tous les concepts et propriétés de la théorie de la complexité, mais uniquement ceux nécessaires à la compréhension des problèmes considérés dans la suite. Pour des informations plus détaillées, nous recommandons aux relecteurs de consulter les ouvrages suivants : [Pap94, PS82, CKS01, CGJ96, eJYGI91].

La théorie de la complexité est une branche de l'informatique théorique qui étudie la difficulté intrinsèque des problèmes de calcul. Elle se concentre sur l'analyse des ressources nécessaires pour résoudre ces problèmes, telles que le temps d'exécution et l'espace mémoire. L'objectif principal de la théorie de la complexité est d'établir des limites et des frontières sur la faisabilité de la résolution de certains problèmes, ainsi que de classer les problèmes en fonction de leur degré de difficulté. Elle fournit des outils et des techniques pour évaluer la complexité des algorithmes, prédire leurs performances et identifier les problèmes pour lesquels des solutions efficaces existent.

Plus précisément, la théorie de la complexité vise à étudier l'évolution des ressources requises pour résoudre un problème en fonction de la taille des données d'entrée. En d'autres termes, pour une procédure $\mathcal{P}$, cette théorie s'intéresse à analyser comment le temps ou l'espace nécessaire pour exécuter $\mathcal{P}$ sur une entrée de taille d varie avec d. En règle générale, le temps de calcul est considéré comme la ressource prédominante pour évaluer la complexité d'un algorithme. Ainsi, dans la suite de ce manuscrit, sauf mention contraire, nous utiliserons le terme « complexité » pour désigner spécifiquement la complexité temporelle.

1.2.1 Principes de base et définitions

La complexité temporelle des algorithmes est évaluée en utilisant une définition précise du temps qui ne dépend pas de la vitesse d'exécution de la machine. Il est important de noter que le temps de calcul mesuré lors de l'exécution d'un programme sur une machine donnée ne reflète pas directement la difficulté intrinsèque du problème que l'algorithme résout. La disponibilité d'un nouveau processeur ne rend pas un problème plus facile à résoudre. Afin de mieux estimer la complexité d'un algorithme, il est essentiel de se dégager de cette notion temporelle liée au matériel et de considérer le temps comme le nombre d'opérations élémentaires $\mathcal{T}_{\mathcal{P}}(d)$ requises pour exécuter le programme sur des entrées de taille d.

Définition 4 (notion informelle d'algorithme). *Informellement, un algorithme peut être décrit comme une séquence d'instructions précises pour résoudre un problème spécifique.*

Exemple 6 (recherche dichotomique). *Soit une liste triée de nombres. Nous voulons déterminer si un nombre est présent dans cette liste. Pour cela, il existe plusieurs méthodes, parmi ces méthodes, nous utilisons la recherche dichotomique que nous appelons $\mathcal{D}$, Nous débutons par examiner l'élément au milieu de la liste, nous répétons le processus de comparaison en examinant l'élément au milieu de la*

liste initiale, puis de la moitié supérieure ou inférieure, jusqu'à ce que nous trouvions une l'élément recherché.

- *Le meilleur cas est celui où l'élément recherché est au milieu de la liste, dans ce cas* $\mathcal{T}_{\mathcal{D}}(d) = 1$
- *Le pire cas est celui où l'élément est le dernier ou le premier de la liste, dans ce cas* $\mathcal{T}_{\mathcal{D}}(d) = \lfloor \log_2(d) \rfloor + 1$

En pratique, nous étudions souvent le comportement de l'algorithme dans le pire des cas afin de garantir des performances minimales dans toutes les situations et de faciliter la planification des ressources nécessaires à son fonctionnement.

Définition 5 (mesure de calcul de complexité). *La complexité algorithmique est généralement présentée en utilisant la notation asymptotique, qui donne une approximation mathématique de la croissance de la complexité en fonction de la taille des données d'entrée. La notation la plus couramment utilisée est la notation "grand-O", qui donne un majorant de la complexité dans le pire des cas. Formellement, pour une fonction* $f(d)$ *représentant la complexité d'un algorithme en fonction de la taille* d *des données d'entrée, on peut dire que* $f(d) = O(g(d))$ *si et seulement il existe un réel positif* c *et un entier* d_0 *tels que* $f(d) \leq c.g(d)$ *pour tout* $d \geq d_0$.

Exemple 7. *L'algorithme de recherche dichotomique proposé dans l'exemple 6 a une complexité en* $O(\log_2(d))$ *car il existe une constante* $c = 4$ *et* $d_0 = 2$ *telle que* $\mathcal{T}_{\mathcal{D}}(d) \leq c \cdot \log_2(d)$ *pour tout* $d \geq d_0$.

Dans le but de fournir une estimation générale de la complexité de divers algorithmes, nous présentons dans le tableau 1 les temps de calcul nécessaires à leur exécution, en tenant compte de leur complexité respective. Toutefois, avant d'examiner ces exemples, nous allons mettre l'accent sur les algorithmes de complexité polynomiale, en la définissant et en donnant un exemple, car bon nombre des algorithmes étudiés dans ce mémoire relèvent de cette catégorie.

Définition 6 (algorithme de complexité polynomiale). *Un algorithme* $\mathcal{A}$ *a une complexité polynomiale, si et seulement s'il existe une fonction polynomiale* $f(d) = c.d^k$*, où* c *et* k *sont des constantes, telle que, le temps d'exécution de l'algorithme* $\mathcal{A}$ *sur une entrée de taille* d *soit majoré par* $f(d)$.

Exemple 8. *Le nombre exact d'opérations élémentaires pour l'algorithme de tri par insertion dans le pire des cas est de* $\frac{d^2-d}{2}$*, il est donc admis que le l'algorithme est en* $O(d^2)$.

Pour donner une idée générale des différentes complexités, le tableau ci-après 1 présente pour différentes types de complexité, leurs noms, des temps d'exécution de référence. Les temps d'exécution sont estimés sur la base d'un accès mémoire de 10 nano-secondes par étape. Il est important de noter que ces temps d'exécution n'ont pas de valeur réaliste, car de nombreux mécanismes sont impliqués lors de l'exécution sur une machine réelle. Ils sont fournis à titre indicatif pour donner un ordre de grandeur du temps nécessaire à l'exécution des algorithmes correspondants.

1.2.2 Rappels sur la machine de Turing

Une machine de Turing est un modèle abstrait de calcul qui consiste en une tête de lecture / écriture bougeant le long d'un ruban infini divisé en cases. Chaque case peut contenir un symbole pris dans un ensemble fini. La machine peut se trouver dans un état particulier parmi un ensemble fini d'états. À chaque étape, la machine effectue une transition en fonction de l'état actuel et du symbole lu sur la case actuelle. Elle peut modifier le symbole de la case, changer d'état et déplacer sa tête vers la gauche ou la droite. Ce modèle permet de simuler des algorithmes en manipulant des symboles sur le ruban, offrant ainsi une représentation abstraite des processus de calcul.

Temps	**Type de complexité**	d = 5	d = 10	d = 1,000	d = 10,000	d = 1,000,000
$O(1)$	Complexité constante	10 ns	10 ns	10 ns	10 ns	10 ns
$O(\log(d))$	Complexité logarithmique	10 ns	10 ns	30 ns	40 ns	60 ns
$O(\sqrt{d})$	Complexité racinaire	22 ns	32 ns	158 ns	1 µs	10 µs
$O(d)$	Complexité linéaire	50 ns	100 ns	2.5 µs	100 µs	10 ms
$O(d \log^*(d))$	Complexité quasi linéaire	50 ns	100 ns	2.5 µs	100.5 µs	10.05 ms
$O(d \log(d))$	Complexité linéarithmique	40 ns	100 ns	6 µs	400 µs	60 ms
$O(d^2)$	Complexité quadratique	250 ns	1 µs	625 µs	1 s	2.8 heures
$O(d^3)$	Complexité cubique	1.25 µs	10 µs	156 ms	10 s	2.7 heures
$O(2^{\text{poly}(\log(d))})$	Complexité sous-exponentielle	30 ns	100 ns	5 ms	10 s	3.2 ans
$O(2^{\text{poly}(d)})$	Complexité exponentielle	320 ns	10 µs	10^{59} ans	...	...
$O(d!)$	Complexité factorielle	1.2 µs	36 ms	10^{48} ans	...	...
$O(2^{2^{\text{poly}(d)}})$	Complexité doublement exp	4.3 s	10^{278} ans	...	...	...

TABLE 1 – Estimation des temps d'exécution (ordre de grandeur) en fonction de la complexité des algorithmes et de la taille des données d'entrée.

La thèse de **Church** postule que tout problème de calcul fondé sur une procédure algorithmique peut être résolu par une machine de Turing. Cette thèse n'est pas un énoncé mathématique, puisqu'elle ne suppose pas une définition précise des procédures algorithmiques. En revanche, il est possible de définir une notion de « système acceptable de programmation » et de démontrer que le pouvoir de tels systèmes est équivalent à celui des machines de Turing (ils sont dits Turing-complets).

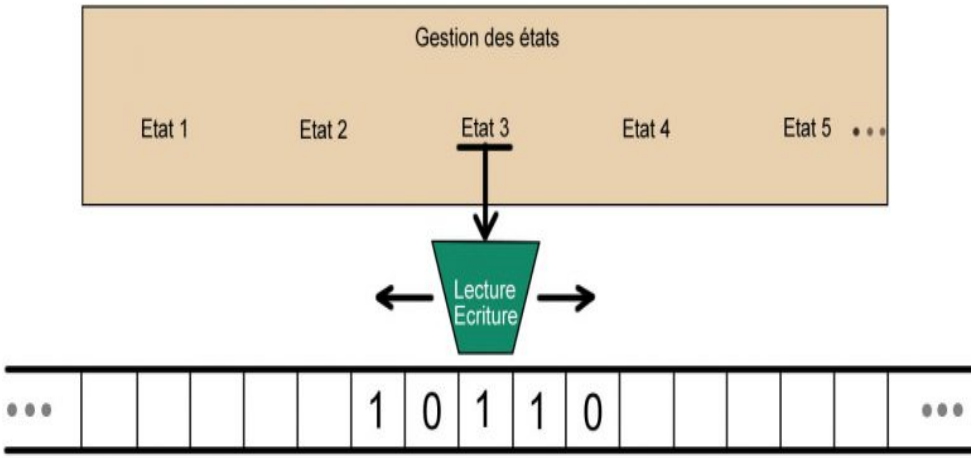


FIGURE 3 – Une machine de Turing (source : turing)

Une machine de Turing effectue un calcul en suivant les étapes suivantes : la donnée d'entrée est placée sur le ruban, la tête de lecture est positionnée sur la première case de la donnée. Ensuite, le programme se déroule en suivant la fonction de transition jusqu'à ce qu'une transition conduise à un état d'arrêt. Une fois le calcul terminé, le résultat peut être lu sur le ruban.

Remarque 1. *Dans certaines situations, la machine peut continuer à exécuter indéfiniment des étapes sans jamais atteindre un état d'arrêt. On dit alors que la machine entre dans une boucle infinie, elle ne produit pas de résultat.*

Il existe trois types principaux de machine de Turing : la machine de Turing déterministe, la machine

de Turing non déterministe et la machine de Turing avec oracle.

— **Machine de Turing déterministe :** il s'agit du type classique de machine de Turing où, à chaque étape de transition, il n'y a qu'une seule action possible en fonction de l'état courant et du symbole lu sur le ruban. Ainsi, le comportement de la machine est déterminé à chaque étape.

— **Machine de Turing non déterministe :** contrairement à la machine de Turing déterministe, une machine de Turing non déterministe peut considérer plusieurs choix possibles à chaque étape de transition.

— **Machine de Turing avec oracle :** ce type de machine de Turing est une extension de modèle de base avec un « oracle ». L'oracle est une entité externe qui peut fournir des réponses à des questions spécifiques en une transition seulement. Cela permet à la machine d'accéder à des connaissances supplémentaires ou à des capacités de calcul spécifiques fournies par l'oracle.

Ces différentes variations des machines de Turing sont utilisées en informatique théorique pour explorer les limites des calculs et étudier la solvabilité des problèmes.

1.2.3 Classes de complexité et hiérarchie polynomiale

Nous allons maintenant décrire comment les machines de Turing peuvent être utilisées pour classer les problèmes en fonction de leur complexité, c'est-à-dire la difficulté inhérente à leur résolution, plutôt que de se concentrer sur la complexité des algorithmes utilisés pour les résoudre. Avant d'aborder ce sujet, il est nécessaire de rappeler brièvement ce qu'est un problème de décision et son complémentaire.

Définition 7 (**problème de décision**)**.** *Un problème de décision ρ est défini par un ensemble de toutes les instances $\mathcal{I}$ possibles, où chaque instance est une entrée pour le problème. Pour chaque instance, le problème de décision consiste à déterminer si l'instance donnée appartient à l'ensemble des instances satisfaisant la propriété « oui » (noté $\mathcal{O}_\rho$) ou « non » (noté $\mathcal{N}_\rho$).*

Le problème complémentaire d'un problème de décision est obtenu en inversant les réponses « oui » et « non », c'est-à-dire : $\mathcal{N}_{\rho^c} = \mathcal{O}_\rho$ et $\mathcal{O}_{\rho^c} = \mathcal{N}_\rho$. ρ^c représente le complémentaire du problème ρ. Dans la suite, nous exposons différentes classes de complexité associées aux problèmes de décision.

Définition 8 (**classe** P)**.** *La classe de complexité* P *comprend tous les problèmes de décision qui peuvent être résolus en un temps polynomial par une machine de Turing déterministe.*

La classe P regroupe les problèmes de décision qui peuvent être résolus efficacement en temps polynomial par un algorithme déterministe. Cependant, il existe d'autres problèmes pour lesquels nous ne connaissons pas d'algorithme polynomial, bien que cela ne signifie pas qu'il n'en existe pas. Pour englober une classe plus vaste de problèmes, nous introduisons la classe NP.

Définition 9 (**classe** NP)**.** *La classe* NP *regroupe les problèmes de décision qui peuvent être résolus en temps polynomial par une machine de Turing non déterministe.*

Les problèmes appartenant à la classe NP (*Non-Déterministe Polynomial*) sont des problèmes de décision pour lesquels la réponse « oui » peut être vérifiée en temps polynomial par un algorithme déterministe. En d'autres termes, il existe un algorithme polynomial capable de vérifier si une solution donnée est une réponse valide au problème. Par conséquent, la classe NP regroupe les problèmes pour lesquels il est facile de vérifier une solution. Il est possible de définir le problème complémentaire de tout problème appartenant à NP.

Définition 10 (CoNP)**.** *La classe* CoNP *regroupe l'ensemble des problèmes de décision dont les problèmes complémentaires sont dans la classe* NP.

Propriété 1. $\mathsf{P} \subseteq \mathsf{NP}$ *et* $\mathsf{P} \subseteq \mathsf{CoNP}$.

La propriété 1 peut être démontrée en notant que toute machine de Turing déterministe peut être considérée comme une machine de Turing non déterministe particulière. La question de savoir si les inclusions sont strictes ou s'il s'agit d'une égalité est un problème ouvert fondamental. On conjecture que $\mathsf{P} \neq \mathsf{NP}$.

Maintenant, nous allons présenter le concept de complétude. La notion de complétude revêt une importance capitale, car elle nous permet de définir l'ensemble des problèmes les plus difficiles au sein d'une classe donnée. En identifiant ces problèmes complets pour une classe, nous pouvons mieux comprendre les limites de cette classe et déterminer si de nouveaux problèmes y appartiennent. La complétude joue un rôle clé dans la hiérarchie des classes de complexité.

Définition 11 (**problème C-difficile**). *Un problème est dit* C*-difficile s'il est au moins aussi difficile que tous les problèmes de la classe* C*. Cela signifie qu'une solution efficace à ce problème permettrait de résoudre efficacement tous les problèmes de la classe* C*.*

Pour parvenir à une classification précise des problèmes, on utilise le concept de réduction fonctionnelle polynomiale. En d'autres termes, la réduction permet de montrer qu'un premier problème est au moins aussi difficile qu'un second problème.

Définition 12 (**réduction fonctionnelle**). *Pour deux problèmes ρ et ρ', une fonction f est une réduction fonctionnelle polynomiale si elle se calcule avec un algorithme polynomial et si, pour chaque instance π de ρ, $f(\pi)$ est une instance positive de ρ' si et seulement si π est une instance positive de ρ.*

Définition 13. *Un problème ρ est dit complet pour une classe de complexité* C *si ρ appartient à* C *et ρ est* C*-difficile.*

La figure 4 illustre les classes de complexité NP et P ainsi que leurs complémentaires, sous les hypothèses habituelles $\mathsf{P} \neq \mathsf{NP}$ et $\mathsf{NP} \neq \mathsf{CoNP}$.

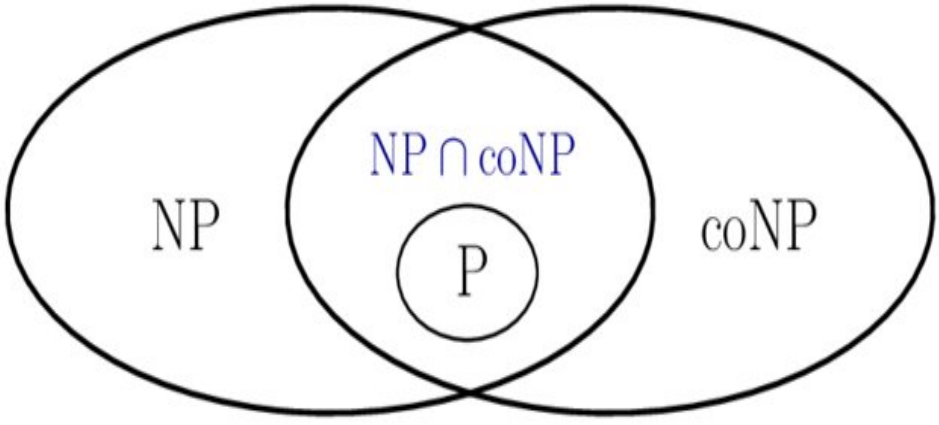


FIGURE 4 – Classes de complexité NP, CoNP (source : [Mos17])

.

Nous allons maintenant introduire la **hiérarchie polynomiale**, une classe de complexité PH défini de manière récursive comme $\mathsf{PH} = \bigcup_{i \geq n} \Sigma_i^{\mathsf{P}}$.

Définition 14 (**hiérarchie polynomiale**). *La hiérarchie polynomiale [Sto76] est une classe de complexité définie de manière récursive à partir de ses sous-classes en utilisant des machines de Turing à oracle, comme suit :*

— $\Delta_0^{\mathsf{P}} = \Sigma_0^{\mathsf{P}} = \Pi_0^{\mathsf{P}} = \mathsf{P}$
— $\Delta_{k+1}^{\mathsf{P}} = \mathsf{P}^{\Sigma_k}$
— $\Sigma_{k+1}^{\mathsf{P}} = \mathsf{NP}^{\Sigma_k}$
— $\Pi_{k+1}^{\mathsf{P}} = \mathsf{Co}\Sigma_{k+1}^{\mathsf{P}}$

Voir par exemple [Pap94] pour plus de détails. Comme nous pouvons le constater sur la figure 5, les problèmes sont classés de manière incrémentale selon leur difficulté croissante. Cette classification se fait en déduisant la classe d'un nouveau problème à partir de celle d'un problème précédent. Cela crée une progression ascendante où chaque nouveau niveau traduit une augmentation de complexité ou de puissance de calcul requise pour résoudre les instances des problèmes de la classe.

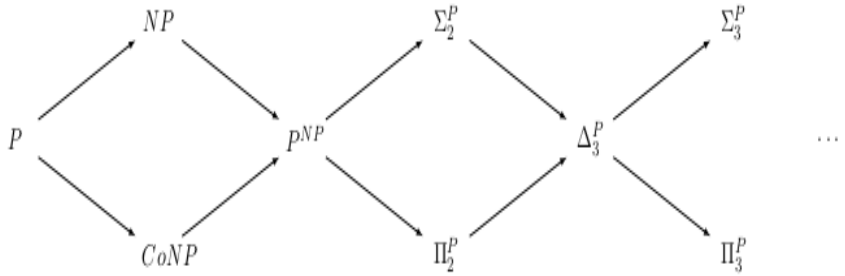


FIGURE 5 – Diagramme de Hasse de la hiérarchie polynomiale (les flèches indiquent l'inclusion entre les classes de complexité).

Pour pouvoir classer des problèmes comme NP-complets, il est essentiel d'identifier un premier problème « initial » qui est NP-complet. Ce premier problème, connu sous le nom de problème de la satisfiabilité propositionnelle (**SAT**), a été démontré comme NP-complet par Cook en 1971 [COO71]. En tant que problème fondamental et emblématique, **SAT** joue un rôle central dans la théorie de la complexité et sert de référence pour évaluer la difficulté des autres problèmes.

1.2.4 Classe de complexité #P

Pour la plupart des algorithmes non déterministes, chaque calcul accepté correspond d'une manière naturelle à une « solution » d'un problème. Il a été largement observé que pour de nombreuses paires de problèmes NP-complets [Val79b], il existe des transformations polynomiales entre elles qui préservent le nombre de solutions. Ces problèmes sont donc équivalents non seulement en ce qui concerne l'existence de solutions mais aussi en ce qui concerne le problème du comptage des solutions. Dans la suite, nous définissons la classe de complexité #P. Cette classe de complexité a été introduite par [Val79a]. Plusieurs problèmes de comptage des solutions appartiennent à cette classe de complexité. Mais avant cela, nous avons besoin de rappeler la définition de machine de Turing de comptage définie également dans [Val79a].

Définition 15 (machine de Turing de comptage [Val79a]). *Une machine de Turing de comptage est une machine de Turing non déterministe standard dotée en plus d'un dispositif de sortie auxiliaire qui imprime en notation binaire sur une bande le nombre de calculs acceptés induits par l'entrée. Elle a une complexité temporelle (dans le pire des cas) $f(n)$ si le plus long calcul d'acceptation induit par l'ensemble de toutes les entrées de taille n prend $f(n)$ étapes (lorsque la machine de Turing est considérée comme une machine non déterministe standard).*

Définition 16 (**classe** $\#\mathsf{P}$ **[Val79a]**). $\#\mathsf{P}$ *est la classe des fonctions qui peuvent être calculées par des machines de Turing de comptage ayant une complexité temporelle polynomiale.*

Une autre caractérisation (souvent donnée comme définition) est la suivante.

Proposition 1. *Une fonction f est dans $\#\mathsf{P}$ si et seulement s'il existe une machine de Turing non déterministe N fonctionnant en temps polynomial telle que pour tout x, $f(x)$ est le nombre de chemins acceptants de $N(x)$.*

1.3 Le problème SAT

Dans cette section, nous abordons différentes formes normales dans le contexte de la logique propositionnelle. Nous nous intéressons à plusieurs problèmes associés à ces formes normales. Plus précisément, nous examinons le problème de la satisfiabilité booléenne, le problème de satisfaction maximale (MAX-SAT) et le du comptage de modèles (# SAT). En outre, nous présentons une forme normale appelée DNF orthogonale.

1.3.1 Formes normales

Les formes normales qui nous intéressent s'appuient sur la notion de terme (déjà introduite) et celle de clause :

Définition 17 (**clause**). *Une clause est une disjonction de littéraux.*

Dans la suite de ce mémoire, nous allons souvent considérer deux formes normales, la forme normale disjonctive (DNF) et la forme normale conjonctive (CNF). Cependant, il est également important de mentionner les formules sous forme normale négative (NNF) afin de comprendre les différentes structures logiques utilisées dans le domaine de la logique propositionnelle.

Définition 18 (**formes normales**). *Nous distinguons les trois formes normales suivantes :*

- DNF *(Forme Normale Disjonctive) : une formule propositionnelle est en forme normale disjonctive si elle est une disjonction de termes, où chaque terme est une conjonction de littéraux.*
- CNF *(Forme Normale Conjonctive) : une formule propositionnelle est en forme normale conjonctive si elle est une conjonction de clauses, où chaque clause est une disjonction de littéraux.*
- NNF *(Forme Normale Négative) une formule propositionnelle est sous forme normale négative si elle est exclusivement constituée de conjonctions, de disjonctions et de littéraux.*

Exemple 9 (**formes normales**). *Voici des exemples de formules en* CNF, DNF *et* NNF *sur un ensemble de variables* $\{a, b, c\}$.

CNF *(Forme Normale Conjonctive) :* $(a \vee \neg b) \wedge (b \vee c) \wedge (\neg a \vee c)$

DNF *(Forme Normale Disjonctive) :* $(a \wedge \neg b) \vee (b \wedge c) \vee (\neg a \wedge c)$

NNF *(Forme Normale Négative) :* $((a \vee \neg b) \vee (b \wedge c)) \wedge (\neg a \vee c)$

Il est clair que les formes normales conjonctives (CNF) et disjonctives (DNF) sont des cas spécifiques de formes normales négatives (NNF). Une formule CNF est dite positive si toutes ses clauses sont positives, c'est-à-dire qu'elles ne contiennent que des littéraux positifs. Une formule CNF est monotone si elle ne contient pas simultanément une variable booléenne et sa négation.

Propriété 2 (**transformation en forme normale conjonctive** CNF). *Toute formule de logique propositionnelle peut être transformée en une formule en forme normale conjonctive équivalente.*

La transformation classique d'une formule propositionnelle vers une formule CNF équivalente peut entraîner une augmentation exponentielle de la taille de la formule. Une approche alternative a été proposée par Tseitin [Tse68]. Cette méthode conduit à introduire des variables supplémentaires dans la formule CNF ψ obtenue, c'est-à-dire des variables qui ne figurent pas dans la formule propositionnelle φ de départ. Les variables supplémentaires de ψ sont définies à partir des variables de départ de φ, ainsi ψ et φ ont les mêmes conséquences logiques sur les variables de départ et le même nombre de modèles sur les variables de départ. La transformation de Tseitin est efficace (elle s'effectue en temps linéaire).

1.3.2 Problème de satisfiabilité booléenne (SAT)

En informatique théorique, le problème **SAT** ou problème de satisfaisabilité booléenne est le problème de décision, qui, étant donné une formule de logique propositionnelle, détermine s'il existe une affectation des variables propositionnelles qui rend la formule vraie.

Le problème **SAT** joue un rôle crucial en théorie de la complexité. Il a été mis en évidence par le théorème de Cook [COO71], qui constitue le fondement de la NP-complétude et de la question P = NP. En plus de son importance théorique, le problème SAT trouve de nombreuses applications pratiques, telles que la satisfaction de contraintes, la planification classique, la vérification de modèles, le diagnostic, et même dans des domaines tels que la configuration d'un ordinateur ou de son système d'exploitation. Dans ces différents domaines, on peut réduire les problèmes à des formules propositionnelles et utiliser un solveur SAT pour obtenir une solution. Les avancées en **SAT** et ses variantes a conduit également à des avancées dans les domaines tels que l'optimisation combinatoire.

Définition 19 (le problème SAT). *Le problème SAT est un problème de décision qui consiste à déterminer si une formule propositionnelle ϕ en forme* CNF *possède un modèle ou non.*

Le problème **SAT** est un problème NP-complet. Pour résoudre ce problème, les meilleurs algorithmes connus ont une complexité exponentielle dans le pire des cas. Malgré cette complexité, des algorithmes sont souvent considérés comme efficaces en pratique pour résoudre les instances de problèmes NP-complets.

Exemple 10 (problème SAT). *Considérons la formule propositionnelle suivante en forme normale conjonctive (*CNF*) :* $(a \vee \neg b) \wedge (b \vee c) \wedge (\neg a \vee c)$.
En examinant la formule, nous pouvons voir qu'il existe plusieurs affectations possibles qui rendent la formule vraie. Par exemple, l'affectation z telle que : $\{z[a] = 1, z[b] = 1, z[c] = 1\}$.

1.3.3 Problème de satisfiabilité maximale (MaxSAT)

Dans cette section, nous introduisons le le problème de satisfaisabilité maximale (MaxSAT) qui est une généralisation du problème SAT. Nous présentons également un abrégé des techniques de résolution mises en œuvre dans les solveurs MaxSAT modernes. Nous définissons les concepts de base, ainsi que les extensions du problème MaxSAT connues sous le nom de Weighted MaxSAT, Partial MaxSAT et Weighted Partial MaxSAT.

Préliminaires. Dans le cadre de MaxSAT, une formule CNF est considérée comme un multi-ensemble de clauses, car les clauses répétées ne peuvent pas être réduites à une unique clause. Par exemple, le multi-ensemble $\{x_1, \neg x_1, \neg x_1, x_1 \vee x_2, \neg x_2, x_1 \vee x_3\}$, où la clause $\neg x_1$ est répétée, tel que toute affectation viole au moins deux clauses. En revanche, si nous considérons l'ensemble $\{x_1, \neg x_1, x_1 \vee x_2, \neg x_2, x_1 \vee$

$x_3\}$, où il n'y pas de répétition de la clause $\neg x_1$, toute affectation viole au moins une clause (et pas deux).

Une clause pondérée est définie comme une paire (C, w), où C représente une clause et w son poids. Le poids peut être un nombre entier ou l'infini ∞. Il représente la pénalité (coût) associée à la falsification de la clause C. Une clause est qualifiée de dure (*hard*) si son poids est l'infini ∞, sinon, elle est qualifiée de souple (*soft*). Une formule CNF pondérée (noté φ) est un multi-ensemble de clauses pondérées, noté $\varphi = \{(C_1, w_1), \cdots, (C_m, w_m)\}$.

Définition 20 (problème de satisfaction maximale (MaxSAT)). *Étant donnée une formule* CNF ϕ *et un entier k, le problème MaxSAT consiste à déterminer s'il existe une affectation satisfaisant au moins k clauses de ϕ. Le problème d'optimisation associé cherche à déterminer une affectation satisfaisante le plus de clauses possible. En conséquence, le problème d'optimisation associé est* NP*-difficile.*

Propriété 3 (complexité du problème MaxSAT). *Le problème MaxSAT est un problème* NP*-complet.*

La preuve de la propriété 3 vient directement du fait que le problème MaxSAT est une généralisation de SAT, SAT est obtenu en fixant k au nombre de clauses de ϕ. Étant donné que SAT est NP-complet, et que MaxSAT est une extension de SAT, cela implique que MaxSAT est également NP-difficile.

Exemple 11 (problème MaxSAT). *Considérons une instance MaxSAT φ avec les clauses suivantes :*

$\varphi = \{x_1 \vee x_2, \neg x_2, \neg x_1 \vee \neg x_2, \neg x_1 \vee x_3, \neg x_1 \vee x_2, \neg x_3\}$

Une solution optimale pour l'instance φ du problème d'optimisation MaxSAT est l'affectation z telle que $\{z[x_1] = 0, z[x_2] = 1, z[x_3] = 0\}$, qui satisfait 5 clauses et ne satisfait pas $\neg x_2$.

Nous allons maintenant définir plusieurs extensions du problème MaxSAT qui sont particulièrement adaptées à la représentation et à la résolution de problèmes sur-contraints : Weighted MaxSAT, Partial-MaxSAT et Weighted Partial MaxSAT.

Définition 21 (Weighted MaxSAT, Partial-MaxSAT et Weighted Partial MaxSAT).

Soit $\varphi = \{(c_1, w_1), \cdots, (c_m, w_m)\}$, où $\forall j \in [m], w_j \in \mathrm{N}^*$, et chaque $c_j (j \in [m])$ est clause. Le problème d'optimisation ***Weighted MaxSAT*** *pour φ consiste à déterminer une affectation qui maximise la somme des poids associés aux clauses satisfaites de ϕ. Formellement, une affectation z qui maximise la somme : $\sum_{j=1}^{m} z[c_j] w_j$.*

Soit la CNF *pondérée $\varphi = \{(c_1, 1), \cdots, (c_m, 1), (c_{m+1}, \infty), \cdots, (c_{m+p}, \infty)\}$. Le problème d'optimisation* ***Partial MaxSAT*** *pour φ est le problème qui consiste à déterminer une affectation qui satisfait toutes les clauses dures (*hard*) et un nombre maximal de clauses souples (*soft*) de φ.*

Soit la CNF *pondérée $\varphi = \{(c_1, w_1), \cdots, (c_m, w_m), (c_{m+1}, \infty), \cdots, (c_{m+p}, \infty)\}$. où $\forall j \in \{1, \cdots, m\}$, $w_j \in \mathbb{N}^*$. Le problème d'optimisation* ***Weighted Partial MaxSAT*** *consiste à déterminer une affectation qui satisfait toutes les clauses dures et maximise la somme des poids associés aux clauses souples satisfaites.*

Remarque 2. *Le problème Partial MaxSAT peut être défini comme Weighted Partial MaxSAT restreint aux formules dont les clauses souples ont un poids de 1. Remarquons aussi que le problème d'optimisation associé à SAT est équivalent à Partial MaxSAT lorsque qu'il n'y a pas de clauses souples.*

Exemple 12 (Weighted Partial MaxSAT). *Considérons la formule* CNF *:*
$\varphi = \{(c_1, 3), (c_2, 1), (c_3, \infty), (c_4, 2), (c_5, \infty), (c_6, 2)\}$ *où les clauses et leurs poids associés sont :*

$c_1 = x_1 \vee x_2, w_1 = 3$

$c_2 = \neg x_2, w_2 = 1$

$c_3 = \neg x_1 \vee \neg x_2, w_3 = \infty$

$c_4 = \neg x_1 \vee x_2, w_4 = 2$

$c_5 = \neg x_3, w_5 = \infty$

$c_6 = \neg x_1 \vee x_3, w_6 = 2$

*Dans cet exemple, les clauses c_3 et c_5 sont des clauses dures (*hard*) avec des poids infinis, tandis que les clauses c_1, c_2, c_4 et c_6 sont des clauses souples (*soft*) avec des poids respectifs de 3, 1, 2 et 2. L'affectation z telle que $\{z[x_1] = 0, z[x_2] = 1, z[x_3] = 0\}$ est une solution de cette instance du problème d'optimisation Weighted Partial MaxSAT. Avec cette affectation, toutes les clauses dures sont satisfaites (clauses c_3 et c_5) et seule la clause souple c_2 reste non satisfaite. La somme des poids associés aux clauses satisfaites est égale à 7.*

Solveurs MaxSAT. Présentons maintenant une brève liste de solveurs MaxSAT qui ont été développés ces dernières années. Le tableau 2 contient la liste des solveurs, les types de problèmes qu'ils traitent (u : MaxSAT non pondéré, w : MaxSAT pondéré, p : MaxSAT partiel, wp : MaxSAT partiel pondéré) ainsi que les noms des auteurs.

Solveur	**Catégories**	**Auteur(s)**
MiniMaxSat	(u, w, p, wp)	Heras et al.
MaxHS	(u, w, p, wp)	Malik et al.
SAT4J-Maxsat	(u,w,p,wp)	Le Berre et al.
QMaxSat	(p)	Koshimura et al.
Open-WBO	(w, wp)	Martins et al.
RC2	(u, w)	Marques-Silva et al.
WBO	(w)	Manquinho et al.

TABLE 2 – Solveurs MaxSAT avec catégories et auteurs

1.3.4 Problème de comptage de modèles (# SAT)

Le problème de comptage de modèles (# SAT) est le problème du comptage du nombre d'affectations qui satisfont une formule donnée. Ce problème a été introduit par Valiant en 1979 [Val79b]. Il s'agit de déterminer le nombre de modèles d'une formule booléenne. Par exemple, pour la formule $a \vee \neg b$, il existe trois affectations distinctes des valeurs booléennes des variables qui la satisfont : ($a = 1$, $b = 0$), ($a = 0$, $b = 0$) et ($a = 1$, $b = 1$). Donc, il existe 3 modèles en tout.

Le problème #SAT se distingue du problème de satisfaisabilité booléenne (SAT) qui demande simplement s'il existe un modèle pour une formule propositionnelle. Au lieu de cela, #SAT vise à compter tous les modèles possibles pour une formule propositionnelle donnée. #SAT est évidement plus difficile que SAT dans le sens où, une fois que le nombre total de solutions pour une formule est connu, SAT peut être résolu en temps constant. Cependant, l'inverse n'est pas vrai, car connaître qu'une formule a une solution ne nous aide pas à compter toutes ses solutions.

Définition 22 (#SAT). *Le problème de comptage de modèles ou #SAT consiste à déterminer le nombre total d'affectations qui satisfont une formule booléenne donnée ϕ.*

Propriété 4 (**complexité de #SAT**). *Le problème #SAT est #*P*-complet.*

Exemple 13 (**comptage de modèles**). *Considérons la formule suivante en forme normale conjonctive (*CNF*) :* $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$. *Le nombre de modèles de cette formule est de* 2. *Les deux assignations qui satisfait la* CNF *précédente sont :* $\{m[x_1] = 1, m[x_2] = 1\}$ *et* $\{m[x_1] = 0, m[x_2] = 0\}$.

Remarque 3. *Il est important de noter que pour une formule propositionnelle donnée* ϕ *sur un ensemble de variables* X_d*, le nombre de modèles de* ϕ *ne peut pas être supérieur à* 2^d.

Contrairement à SAT, le problème de comptage du modèle reste difficile pour les formules DNF. Cependant, il existe certaines sous-familles de formes normales DNF où le comptage de modèles peut être réalisé en temps linéaire. C'est le cas des DNF orthogonales.

1.3.5 DNF orthogonale

Définition 23 (DNF **orthogonale**). *Une* DNF $\phi = t_1 \vee \cdots \vee t_m$ *est orthogonale si et seulement si* $\forall i, j \in [m]$ *si* $i \neq j$ *alors* $t_i \wedge t_j \models \bot$.

La définition 23 stipule simplement que chaque paire de termes d'une DNF orthogonale doit être en conflit sur au moins une variable, c'est-à-dire, il doit y avoir une variable qui apparaît complémentée dans l'un des termes et non complémentée dans l'autre terme. Cette propriété est facile à vérifier pour toute DNF donnée.

Exemple 14 (DNF **orthogonale**). *Soit la formule* $\phi = (\neg x_1 \wedge \neg x_2 \wedge x_4) \vee (\neg x_1 \wedge x_3 \wedge x_4)$. *Cette* DNF *n'est pas orthogonale. Cependant,* ϕ *est équivalente à la* DNF $\psi = (\neg x_1 \wedge \neg x_2 \wedge x_4) \vee (\neg x_1 \wedge x_2 \wedge x_3 \wedge x_4)$, *qui est orthogonale.*

Théorème 1 ([CH11]). *Toute fonction booléenne peut être représentée par une* DNF *orthogonale.*

Bien qu'il soit possible de représenter chaque fonction booléenne par une DNF orthogonale, cette transformation est généralement très coûteuse (en espace exponentiel).

Comptage de modèles et DNF **orthogonale** Dans le cas des DNF orthogonales, le comptage de modèles devient facile. Grâce à la structure spécifique des DNF orthogonales, où chaque terme est orthogonal aux autres, il suffit de compter le nombre de termes évalués à vrai pour obtenir le nombre de modèles de la fonction. Cette propriété des DNF orthogonales facilite grandement le calcul du nombre de modèles.

Proposition 2. *Soit une* DNF *orthogonale* $\phi = t_1 \vee \cdots \vee t_m$ *et qui est définie sur* X_d. *Le nombre de ses modèles est égal à :*

$$w(\phi) = \sum_{k=1}^{m} 2^{d-|t_k|}$$

Exemple 15 (**comptage de modèles**). *Soit* $\psi = (\neg x_1 \wedge \neg x_2 \wedge x_4) \vee (\neg x_1 \wedge x_2 \wedge x_3 \wedge x_4)$ *la* DNF *de l'exemple 14,* ψ *est une* DNF *orthogonale, le nombre de modèles de* ψ *est donné par*

$$w(\psi) = 2^{4-3} + 2^{4-4} = 3.$$

1.4 Conclusion

Dans ce chapitre, nous avons présenté les bases de la logique propositionnelle, un domaine fondamental en mathématiques et en informatique. Les concepts clés que nous avons abordés comprennent les notions d'impliquant et d'impliquant premier. Ces concepts ne sont pas seulement des abstractions théoriques, mais ils jouent un rôle central dans la suite de notre mémoire.

Nous avons également présenté des problèmes qui sont importants dans notre travail. Le problème SAT consiste à déterminer si une formule booléenne est satisfaisable ou non, tandis que le problème d'optimisation MaxSAT vise à déterminer une affectation qui satisfait le plus grand nombre possible de clauses d'une formule `CNF`. Ce dernier problème est plus difficile que le problème SAT. Le troisième problème considéré est le problème du comptage de modèles, ou #SAT, qui consiste à déterminer le nombre de modèles d'une formule donnée. Le problème #SAT est plus difficile (en théorie et en pratique) que les deux problèmes précédents.

2

Optimisation combinatoire

Sommaire

2.1 Introduction 26
2.1.1 Définitions et propriétés clés 27
2.1.2 Problèmes combinatoires NP-complets 28
2.1.3 Matroïdes 29
2.1.4 Optimisation linéaire 31
2.2 Sous-modularité et super-modularité 32
2.2.1 Fonctions sous-modulaires 33
2.2.2 Fonctions super-modulaires 35
2.3 Algorithmes approchés 36
2.3.1 Généralités sur l'approximation 37
2.3.2 Maximisation sous-modulaire 38
2.3.3 Minimisation super-modulaire 39
2.3.4 Importance de la courbure 40
2.4 Conclusion 42

Ce chapitre présente, dans un cadre général, des problèmes d'optimisation combinatoire ainsi que des algorithmes d'approximation pour les résoudre. Nous débutons par présenter des définitions clés et fournir des rappels sur la théorie des matroïdes, la programmation linéaire et les outils nécessaires pour le développement des chapitres suivants, en particulier le chapitre 8. Cela inclut les notions de fonctions sous-modulaires et super-modulaires.

2.1 Introduction

L'optimisation combinatoire est un domaine important de l'optimisation en mathématiques appliquées et en informatique. Il se concentre sur des problèmes d'optimisation où l'ensemble des solutions possibles est fini et souvent de taille exponentielle. Ces problèmes impliquent des décisions discrètes et des contraintes, et ils se retrouvent dans de nombreux domaines, tels que la logistique, la planification, la conception de réseaux, l'ordonnancement, l'affectation des ressources, la bioinformatique, l'apprentissage automatique, etc. L'optimisation combinatoire est étroitement liée à d'autres domaines tels que la recherche opérationnelle, l'algorithmique et la théorie de la complexité. Elle vise à trouver une meilleure solution parmi toutes les solutions possibles, en tenant compte des contraintes et des objectifs spécifiques.

Les problèmes d'optimisation combinatoire sont généralement NP-difficiles, ils peuvent être extrêmement complexes et donc difficiles à résoudre exactement en raison de la taille exponentielle de l'espace de recherche. C'est pourquoi des algorithmes d'approximation sont souvent utilisés pour obtenir des solutions de qualité acceptable dans un temps raisonnable. Des techniques telles que la programmation linéaire, la programmation par contraintes, les méta heuristiques, les algorithmes gloutons, etc., sont largement utilisées dans ce domaine.

2.1.1 Définitions et propriétés clés

Nous supposons que les bases de la théorie de la complexité, telles que présentées au chapitre 1, sont connues des relecteurs. De plus, dans la suite de ce mémoire, nous faisons l'hypothèse que $\mathsf{P} \neq \mathsf{NP}$.

Définition 24 (optimisation combinatoire). *Un problème d'optimisation combinatoire, également connu sous le nom de problème d'optimisation discrète à solution finie, vise à trouver, dans un ensemble discret, l'un des sous-ensembles (ou solutions) réalisables optimaux. La notion d'optimalité est définie par une fonction objectif. En d'autres termes, il s'agit de choisir la meilleure combinaison possible parmi un ensemble limité de choix discrets. Formellement, pour un ensemble discret fini V, une fonction objectif $f : 2^V \to \mathbb{R}$, où 2^V représente l'ensemble des sous-ensembles de V, et $\mathcal{U}$ un ensemble de sous-ensembles de V, il s'agit de déterminer :*

$$\max \{f(S) : S \in \mathcal{U}\} \quad \textit{ou} \quad \min \{f(S) : S \in \mathcal{U}\} \tag{1}$$

À chaque problème d'optimisation, on peut associer un problème de décision dont le but est de déterminer s'il existe une solution pour laquelle la fonction objectif satisfait une contrainte donnée. La complexité d'un problème d'optimisation est liée à celle du problème de décision qui lui est associé. En particulier, si le problème de décision est NP-complet, alors le problème d'optimisation est NP-difficile.

Les problèmes combinatoires en pratique. La théorie de la complexité nous indique que si un problème est NP-difficile, il est peu réaliste de chercher un algorithme polynomial pour le résoudre car il n'en existe pas, à moins que P = NP. Cependant, ces conclusions sont basées sur une évaluation de la complexité dans le pire des cas, en se référant à l'instance la plus difficile du problème. En pratique, bien que nous sachions qu'il est impossible de résoudre en pratique en temps polynomial toutes les instances d'un problème NP-difficile, certaines instances sont en réalité beaucoup plus faciles que d'autres et peuvent être résolues rapidement.

De nombreux problèmes réels sont connus pour être NP-complets ou NP-difficiles. Parmi ces problèmes, on peut citer la conception d'emplois du temps, le problème du sac à dos, l'équilibrage des chaînes de montage, le routage des véhicules. Dans la suite, nous donnons comme exemple le problème du sac-à-dos.

Le problème du sac-à-dos. Prenons comme exemple, le problème du sac-à-dos. Considérons d objets, notés $i = 1, \ldots, d$, apportant chacun un bénéfice c_i mais possédant un poids a_i. On souhaite placer ces objets dans un sac de capacité maximale b. Le problème du sac-à-dos (knapsack) consiste à sélectionner les objets parmi les d disponibles de manière à maximiser le bénéfice total tout en respectant la contrainte de poids maximal autorisé. Chaque objet i, où $i \in [d]$, doit être choisi au moins p_i fois et au plus q_i fois. Ce problème se rencontre bien entendu dès que l'on part en randonnée en voulant emmener le plus possible d'objets utiles ($nourriture$, $boissons$,...). Mais ce problème est plus fréquemment utilisé pour remplir les camions de transport, les avions ou bateaux de fret et même pour gérer la mémoire d'un microprocesseur.

Exemple 16 (le problème du sac-à-dos). *La formulation du problème de sac-à-dos est très simple. On utilise pour chaque objet $i \in \{1, ..., d\}$, une variable entière x_i correspondant au nombre de fois où l'objet i est choisi. Le problème du sac-à-dos est donc équivalent au programme linéaire suivant :*

$$\begin{aligned} \textit{max} \quad & \sum_{i=1}^{d} c_i x_i \\ \textit{s.t.} \quad & \sum_{i=1}^{d} a_i x_i \leq b \\ & p_i \leq x_i \leq q_i, \quad \textit{pour } i = 1, \ldots, d \\ & x_i \in \mathbb{N}, \quad \textit{pour } i = 1, \ldots, d \end{aligned}$$

La contrainte $\sum_{i=1}^{d} a_i x_i \leq b$ est dite contrainte de sac-à-dos. Cette contrainte de cardinalité rend le problème généralement NP*-difficile.*

2.1.2 Problèmes combinatoires NP-complets

De nombreux problèmes réels relèvent de la catégorie des problèmes combinatoires. Dans la pratique, notre objectif est de trouver des solutions à ces problèmes. Étant donné que la théorie de la complexité nous indique qu'il est peu probable de trouver un algorithme qui soit à la fois rapide, correct et complet, nous devons faire des compromis en sacrifiant certains de ces critères. Notre objectif est d'obtenir un comportement *acceptable en pratique*, c'est-à-dire être en mesure de trouver une solution de qualité suffisante dans un délai raisonnable pour les instances de problèmes à résoudre. Pour cela, nous distinguons trois types de méthodes : les méthodes exactes et les méthodes heuristiques et les méthodes approchées. Voici deux familles de **méthodes exactes :**

— **La programmation dynamique :** la programmation dynamique est une méthode exacte utilisée pour résoudre certains problèmes NP-complets dits « faibles ». Cette approche repose sur l'utilisation de techniques récursives et de la mémorisation des résultats intermédiaires pour éviter de recalculer des sous-problèmes déjà résolus. Elle est particulièrement adaptée aux problèmes qui peuvent être formulés de manière récursive ou qui peuvent être décomposés en sous-problèmes de taille plus petite.

— **Les méthodes arborescentes :** une autre approche couramment utilisée pour résoudre des problèmes combinatoires repose sur la construction et l'exploration d'arbres de recherche, où chaque nœud représente une solution partielle. Les méthodes arborescentes permettent d'explorer de manière systématique l'ensemble des solutions possibles en utilisant des stratégies de parcours adaptées, telles que la recherche en profondeur ou la recherche en largeur. Elles permettent d'énumérer implicitement toutes les solutions du problème.

Il est important de noter que ces deux familles de méthodes, la programmation dynamique et les méthodes arborescentes, sont des approches exactes qui garantissent de trouver une meilleure solution possible. Cependant, leur complexité en pratique peut augmenter rapidement avec la taille de l'instance considérée, ce qui limite leur applicabilité aux instances de petite ou moyenne taille. Dans certains cas, même si ces méthodes sont exactes, elles peuvent être trop coûteuses en termes de temps de calcul pour des instances plus grandes. Leur résolution peut alors nécessiter l'utilisation de méthodes heuristiques.

— **Méthode gloutonne :** la méthode gloutonne est une approche heuristique qui consiste à prendre localement la meilleure décision à chaque étape, sans tenir compte des conséquences à long terme. À chaque étape, l'algorithme choisit une meilleure option disponible selon un critère spécifique, souvent basé sur une fonction d'évaluation. Bien que cette méthode puisse conduire à des solutions rapides, elle ne garantit pas de déterminer une meilleure solution globale, car elle peut se retrouver piégée dans des optimums locaux.

— **Recuit simulé :** le recuit simulé est une méthode heuristique inspirée de la métallurgie, où un matériau est chauffé puis progressivement refroidi pour atteindre un état d'équilibre. Dans le contexte de l'optimisation, le recuit simulé utilise une fonction de coût et explore l'espace des solutions en acceptant parfois des mouvements conduisant à dégrader la valeur valeur atteinte. Cette exploration permet d'éviter de rester piégé dans des optimums locaux et d'atteindre potentiellement de meilleures solutions globales.

— **Liste tabou :** la méthode de la liste tabou est une approche heuristique qui utilise une mémoire à court terme pour éviter de revisiter des solutions déjà explorées. L'algorithme maintient une liste de mouvements interdits, appelée liste tabou, et évite de les répéter. Cela permet d'explorer davantage l'espace des solutions et d'éviter de rester bloqué dans des cycles. La liste tabou peut être adaptée en fonction des mouvements effectués et des améliorations observées pour guider la recherche vers de meilleures solutions.

— **Algorithmes génétiques :** les algorithmes génétiques sont des méthodes heuristiques inspirées de la théorie de l'évolution et de la génétique. Ils utilisent des opérations de sélection, de croisement et de mutation pour explorer l'espace des solutions et rechercher des solutions optimales. Les solutions sont représentées sous forme de chromosomes, qui sont combinés et modifiés pour générer de nouvelles solutions. Les algorithmes génétiques peuvent être efficaces pour les problèmes combinatoires complexes, mais leur performance dépend de la représentation des solutions et des opérateurs génétiques utilisés.

Ces méthodes heuristiques ne garantissent pas de fournir une solution optimale. Elles offrent des approches alternatives à la recherche exhaustive de solutions optimales et sont souvent utilisées pour résoudre des problèmes complexes où les méthodes exactes sont trop coûteuses en termes de temps de calcul. Elles sont adaptées à différentes situations et exigent une fine compréhension du problème à résoudre pour choisir la méthode appropriée. Nous allons maintenant définir les problèmes NP-complets au sens fort.

Définition 25. *Un problème est dit* NP*-complet au sens fort s'il est à la fois* NP*-complet et si l'existence d'un algorithme pseudo-polynomial pour le résoudre implique l'existence d'un algorithme polynomial pour résoudre tous les problèmes de la classe* NP*.*

Exemple 17. *Les problèmes du voyageur de commerce et du satisfiabilité (SAT) sont des problèmes* NP*-complets au sens fort.*

2.1.3 Matroïdes

Le but de cette section est d'introduire la théorie des matroïdes en présentant leurs principales propriétés et leurs liens avec l'optimisation. Une compréhension de base de la théorie des graphes et de l'algèbre linéaire est requise pour les exemples et les applications. Le terme « matroïde » est apparu pour la première fois dans l'article fondateur de H. Whitney en 1935 [Whi35]. Dans cet article, la structure de matroïde est introduite comme une abstraction ensembliste des relations de dépendance entre les vecteurs colonnes d'une matrice, d'où le suffixe "-oïde" qui indique qu'il s'agit d'une structure similaire à une

matrice. Pour une étude approfondie de cette théorie, nous recommandons le livre « Matroid theory » de J. Oxley [Oxl06].

Nous rappelons que l'indépendance linéaire d'une famille de vecteurs correspond au fait qu'aucun des vecteurs ne peut être exprimé comme une combinaison linéaire des autres. Les matroïdes cherchent à généraliser cette notion d'indépendance.

Définition 26 (matroïde). *Un matroïde M est défini comme un couple (E, I), où E est un ensemble fini et $\mathcal{I}$ est une famille de sous-ensembles de E qui satisfait les conditions suivantes :*

- $\varnothing \in \mathrm{I}$
- *Si $I_1 \in \mathrm{I}$ et $I_2 \subseteq I_1$, alors $I_2 \in \mathcal{I}$*
- *(Propriété d'augmentation) Si $I_1, I_2 \in \mathrm{I}$ et $\lvert I_1\rvert < \lvert I_2\rvert$, alors il existe un élément $e \in I_2 \setminus I_1$ tel que $I_1 \cup \{e\} \in \mathrm{I}$*

Les ensembles appartenant à I *sont appelés « indépendants » de M. Un sous-ensemble de E qui n'appartient pas à* I *est appelé « dépendant ».*

Exemple 18 (matroïde). *Considérons l'ensemble des sous-ensembles d'un ensemble fini $E = \{1, 2, 3\}$. La famille des indépendants* I *est définie comme l'ensemble de tous les sous-ensembles de E. Par exemple $\mathrm{I} = \{\{\varnothing\}, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$. Le couple (E, I) est un matroïde car il vérifie les trois conditions de la définition 26.*

Définition 27 (rang de matroïde). *Soit $M = (E, \mathrm{I})$ un matroïde et $X \subseteq E$. Le rang de X, noté $r_M(X)$, correspond au nombre maximal d'éléments indépendants contenus dans X, c'est-à-dire $r_M(X) = \max\{\lvert Y\rvert : Y \subseteq X, Y \in \mathrm{I}\}$. On peut également définir l'ensemble $\mathrm{I} \setminus X = \{I \subseteq X : I \in \mathrm{I}\}$. Alors $(X, \mathrm{I} \setminus X)$ forme un matroïde, noté $M|X$ et appelé restriction de M à X. Le rang $r_M(X)$ de X correspond au cardinal d'une base de $M|X$.*

Nous pouvons démontrer que $r = r_M$ est une fonction de rang d'un matroïde (E, I) si et seulement si elle satisfait les conditions suivantes :

- $0 \leq r(X) \leq \lvert X\rvert$ pour tout $X \subseteq E$
- $r(X) \leq r(Y)$ pour tout $X \subseteq Y$
- (inégalité sous-modulaire) $r(X \cup Y) + r(X \cap Y) \leq r(X) + r(Y)$ pour tout $X, Y \subseteq E$.

La condition finale est typique des fonctions sous-modulaires. Un matroïde est une fonction de ce type avec une croissance unitaire (l'ajout d'un élément augmente le rang d'au plus un). Dans le cas vectoriel, le concept de rang correspond à celui de l'algèbre linéaire. Dans le contexte des graphes, le rang est équivalent au nombre de sommets moins le nombre de composantes connexes.

Algorithme glouton du matroïde. Les matroïdes ont des applications étendues dans le domaine de l'optimisation combinatoire, offrant des outils puissants pour résoudre divers problèmes. L'une de leur applications les plus remarquable est l'utilisation des matroïdes pour concevoir des algorithmes gloutons efficaces. L'algorithme glouton d'un matroïde consiste à construire une solution optimale de manière incrémentale en en sélectionnant de manière gourmande des éléments indépendants en suivant une stratégie particulière. Cette stratégie est basée sur la notion de rang du matroïde, qui permet de déterminer quels éléments peuvent être ajoutés sans violer les contraintes d'indépendance. Formellement, soit I un ensemble de sous-ensembles de E et $M = (E, \mathrm{I})$ un matroide. Soit $w : E \to \mathbb{R}$ une fonction de poids (on dit que $w(e)$ est le poids de l'élément e). On définit $w(X) = \sum_{x \in X} w(x)$ pour $X \in E$, et $w(\varnothing) = 0$.

Remarque 4. *L'algorithme glouton du matroïde n'est pas une approche heuristique (au sens large) : c'est un algorithme optimal pour le problème de minimisation / maximisation linéaire dès lors que*

Algorithm 1 Algorithme glouton du matroïde

Input: un matroïde $M = (E, \mathrm{I})$, une fonction de poids w
Output: un sous-ensemble S de E
/*Le problème ici consiste à trouver $S \subseteq E$ qui maximise $w(S)$ */
$S_0 \leftarrow \varnothing$
$j \leftarrow 0$
while $\exists e \in E \setminus S_j$ *such that* $S_j \cup \{e\} \in \mathrm{I}$ **do**
  $e^* \leftarrow \underset{e \in E}{\operatorname{argmax}}\, w(S_j \cup \{e\})$
  $S_{j+1} \leftarrow S_j \cup \{e^*\}$
  $j \leftarrow j + 1$
return S_j

notre espace de solutions $\mathcal{U}$ est un matroïde. Donc on ne confond pas l'algorithme glouton du matroide « Greedy Matroid Algorithm » avec un algorithme glouton qui s'inspire de lui pour résoudre des problèmes d'optimisation non-linéaires (dans ce cas, c'est bien en général une méthode heuristique).

2.1.4 Optimisation linéaire

Introduction. Dans le domaine de l'optimisation combinatoire, il existe différents types de problèmes. L'un de ces problèmes est l'optimisation linéaire, où l'objectif est de maximiser ou minimiser une fonction linéaire sous des contraintes linéaires. La programmation linéaire consiste à formuler un problème d'optimisation linéaire sous une forme mathématique spécifique. L'objectif est de trouver les valeurs des variables (appelées variables de décision) qui optimisent la fonction objectif tout en satisfaisant les contraintes linéaires. La particularité de la programmation linéaire réside dans le fait que des méthodes spécifiques, telles que le simplexe ou les méthodes de points intérieurs, peuvent être utilisées pour résoudre efficacement ces problèmes. Ces méthodes exploitent la structure linéaire du problème pour trouver une solution optimale en un temps raisonnable, même pour des problèmes de grande taille. Il est donc important de distinguer la programmation linéaire des autres problèmes d'optimisation combinatoire, car elle bénéficie de techniques spécifiques qui permettent une résolution efficace. Cela signifie que pour les problèmes qui peuvent être formulés comme des problèmes de programmation linéaire, il existe des outils et des algorithmes dédiés qui peuvent être utilisés pour trouver rapidement une solution optimale ou proche de l'optimalité.

Matroïdes et programmation linéaire. Les matroïdes sont étroitement liés à la programmation linéaire. En effet, de nombreux problèmes d'optimisation linéaire peuvent être formulés en termes de matroïdes. Plus précisément, les matroïdes permettent de définir des ensembles indépendants et des bases, qui sont des concepts clés en programmation linéaire. Les bases d'un matroïde peuvent être des solutions admissibles dans un problème de programmation linéaire, tandis que la fonction rang du matroïde correspond à la valeur optimale de la fonction objectif. L'algorithme glouton du matroïde 1, présenté à la section précédente, est un algorithme utilisé pour résoudre des problèmes d'optimisation combinatoire basés sur les matroïdes.

Définition 28 (**programme linéaire**). *Un programme linéaire **(PL)** est la description d'un problème qui consiste à maximiser ou minimiser une fonction linéaire f de $\mathbb{R}^d$ dans $\mathbb{R}$ sous des contraintes linéaires.*

Formellement, pour une fonction sous la forme suivante :

$$\begin{aligned} \textit{max} \quad & f(x) \\ \textit{s.t} \quad & Ax \leq b \\ & x \in \mathbb{R}^d, \;\; A \in \mathbb{R}^{n \times d}, \;\; b \in \mathbb{R}^n \end{aligned}$$

Exemple 19. *Soit une fonction linéaire $f : \mathbb{R}^2 \longrightarrow \mathbb{R}$, où $f(x_1, x_2) = -2x_1 + 3x_2$, nous définissons le problème d'optimisation linéaire suivant :*

$$\begin{aligned} \textit{max} \quad & f(x_1, x_2) \\ \textit{s.t} \quad & x_1 \geq 0 \\ & 2x_2 - x_1 \leq 2 \\ & x_2 - 2x_1 \geq -4 \\ & x_1 + x_2 \geq 0 \end{aligned}$$

La solution du problème défini dans l'exemple 19 est un vecteur dans $\mathbb{R}^2$.

Présentons brièvement quelques techniques qui permettent de résoudre les programmes linéaires.

— **méthode graphique :** la méthode graphique est une approche graphique utilisée pour résoudre les problèmes de programmation linéaire (PL) ayant un nombre limité de variables. Elle consiste à représenter les contraintes du problème sous forme de droites ou de plans dans un espace à deux ou trois dimensions. Ensuite, on identifie la région de faisabilité en superposant les contraintes et on recherche le point optimal en utilisant les propriétés géométriques de cette région. Cette méthode est généralement restreinte aux problèmes ayant un petit nombre de variables, car la visualisation graphique devient complexe lorsque le nombre de variables augmente.

— **méthode des sommets :** la méthode des sommets, également appelée méthode des points extrêmes, consiste à trouver la solution optimale en évaluant les points extrêmes de la région de faisabilité. Chaque point extrême est défini comme une intersection de plusieurs contraintes du problème. Cette méthode peut être très coûteuse en termes de temps de calcul, car le nombre de sommets de la région de faisabilité peut être prohibitif, en particulier pour les problèmes avec un grand nombre de variables.

— **méthode du simplexe :** la méthode du simplexe est un algorithme itératif développé par George Dantzig en 1951 pour résoudre les problèmes de programmation linéaire. C'est l'une des méthodes les plus couramment utilisées pour résoudre les problèmes de programmation linéaire de manière efficace. L'algorithme commence par un point initial et itère à travers une séquence de points qui améliorent progressivement la valeur de la fonction objectif. À chaque itération, le simplexe se déplace d'un sommet à un autre sommet adjacent jusqu'à ce qu'il atteigne un sommet optimal qui satisfait toutes les contraintes du problème. L'avantage de la méthode du simplexe est sa capacité à gérer des problèmes avec un grand nombre de variables et de contraintes, ce qui la rend largement applicable.

2.2 Sous-modularité et super-modularité

La sous-modularité et la super-modularité sont une propriété des fonctions d'ensembles avec des conséquences théoriques profondes et des applications étendues. Elles apparaissent dans une grande variété d'applications telles que le marketing viral [KKT03], la collecte d'informations [KG07], la segmentation d'images [KKT09], le résumé de documents [LB11] et l'accélération des solveurs de satisfiabilité

booléenne (SAT) [SG08]. Dans cette étude, nous présentons la notion de la sous-modularité et de super-modularité ainsi que certaines de leurs généralisations, et nous illustrons comment elles se manifestent dans diverses applications. [2]

2.2.1 Fonctions sous-modulaires

La sous-modularité est une propriété des fonctions d'ensemble, c'est-à-dire des fonctions $f : 2^V \to \mathbb{R}$ qui assignent à chaque sous-ensemble $S \subseteq V$ une valeur $f(S)$. Ici, V est un ensemble fini, et $f(S)$ représente la valeur de la fonction sur $\mathbb{R}$. La sous-modularité peut être définie de deux manières équivalentes, que nous allons décrire maintenant. La première définition repose sur une notion de dérivée discrète, souvent appelée gain marginal.

Définition 29 (dérivée discrète). *Pour une fonction d'ensemble $f : 2^V \to \mathbb{R}$, $S \subseteq V$, et $e \in V$, la dérivée discrète de f à S au point e (notée $G_f(e|S)$) est définie par $G_f(e|S) = f(S \cup e) - f(S)$. Lorsque la fonction f est claire dans le contexte, nous écrivons simplement $G_f(e|S)$.*

Définition 30 (sous-modularité [Sat05]). *Une fonction $f : 2^V \to \mathbb{R}$ est sous-modulaire si pour tout $A \subseteq B \subseteq V$ et $e \in V \setminus B$, on a $G_f(e|A) \geq G_f(e|B)$. De manière équivalente, une fonction $f : 2^V \to \mathbb{R}$ est sous-modulaire si pour tout $A, B \subseteq V$, nous avons : $f(A \cap B) + f(A \cup B) \leq f(A) + f(B)$.*

Nous avons besoin de ces définitions principalement pour la maximisation sous-modulaire. L'intuition fournie par la première définition est souvent utile. Supposons que nous interprétions $S \subseteq V$ comme un ensemble d'actions visant à maximiser une certaine fonction $f(S)$. Alors, la première définition stipule que pour une fonction sous-modulaire f, une fois que nous avons effectué un ensemble A d'actions, le bénéfice marginal de toute action e ne s'accroît pas lorsque nous effectuons les actions dans $B \setminus A$. Ainsi, les fonctions d'ensemble sous-modulaires présentent une propriété intrinsèque de rendements décroissants.

Exemple 20 (sous-modularité). *Supposons que nous ayons une collection d'ensembles finis $\{S_i : i \in V\}$. Pour $A \subseteq V = \{1, \cdots, d\}$, définissons*

$$S_A = \bigcup_{i \in A} S_i, \quad f(A) = |S_A|,$$

où $|\cdot|$ représente la cardinalité d'un ensemble. En d'autres termes, $f(A)$ est le nombre d'éléments couverts par les ensembles indexés par A. Il est bien clair que fonction est positive et croissante, $f(A)$ est sous-modulaire car pour tout $A \subseteq B$ et $i \notin B$, $f(A \cup \{i\}) - f(A) = |S_i \setminus S_A|$, tandis que $f(B \cup \{i\}) - f(B) = |S_i \setminus S_B|$. Cette dernière quantité ne peut pas être supérieure à la première, car $S_B \supseteq S_A$.

Une classe importante de fonctions sous-modulaires est celle des fonctions croissantes (resp. décroissantes), où l'ajout d'éléments à l'ensemble d'arguments n'entraîne pas une diminution (resp. une augmentation).

Définition 31 (monotonie). *Une fonction $f : 2^V \to \mathbb{R}$ est dite croissante si, pour tout $A, B \subseteq V$ tels que $A \subseteq B$, nous avons $f(A) \leq f(B)$. f est décroissante si, pour tout $A, B \subseteq V$ tels que $A \subseteq B$, nous avons $f(A) \geq f(B)$. Une fonction est dite monotone si elle est croissante sur tout son domaine, ou décroissante sur tout son domaine.*

2. Dans cette section, nous supposons que le lecteur est à l'aise avec les notions de concavité et de convexité.

Notons également que pour une fonction f, la propriété de monotonie est équivalente à ce que toutes ses dérivées discrètes soient positives (ou toutes négatives). Cela signifie que pour tout $A \subseteq V$ et $e \in V$, nous avons $G_f(e|A) \geq 0$ ($G_f(e|A) \leq 0$). La sous-classe des fonctions sous-modulaires croissantes peut être caractérisée en exigeant que pour tout $A \subseteq B \subseteq V$ et $e \in V$, nous ayons $G_f(e|A) \geq G_f(e|B)$. Cette caractérisation diffère légèrement de la définition 30 en ce sens que nous n'exigeons pas que $e \notin B$.

Propriétés des fonctions sous-modulaires. Les fonctions sous-modulaires possèdent de nombreuses propriétés. Nous présentons ici certaines propriétés qui sont utiles dans la suite de ce mémoire.

Un point important lorsque nous travaillons dans le contexte des fonctions d'ensemble est de savoir si une classe de fonctions est préservée par combinaison linéaire. Dans le cas de la sous-modularité, celle-ci n'est pas nécessairement préservée lorsqu'on effectue des combinaisons linéaires arbitraires, à moins que la combinaison des fonctions ne soit conique [3].

Proposition 3. *La sous-modularité n'est pas nécessairement préservée par combinaison linéaire arbitraire. Formellement, si $f_1, \ldots, f_d : 2^V \to \mathbb{R}$ sont des fonctions sous-modulaires et $\lambda_1, \ldots, \lambda_d$ des constantes réelles, alors la fonction $f(S) = \sum_{i=1}^{d} \lambda_i f_i(S)$ n'est pas nécessairement sous-modulaire.*

La proposition 3 peut être démontrée facilement. En effet, il suffit de prendre $d = 1$ et $\lambda_1 < 0$. Cependant, si nous imposons $\lambda_i \geq 0$ (pour tout i), alors la combinaison linéaire positive de fonctions sous-modulaires reste sous-modulaire.

Propriété 5. *La sous-modularité est préservée par combinaison linéaire positive. Formellement, si $f_1, \ldots, f_d : 2^V \to \mathbb{R}$ sont des fonctions sous-modulaires et $\lambda_1, \ldots, \lambda_d \geq 0$, alors la fonction $f(S) = \sum_{i=1}^{d} \lambda_i f_i(S)$ est également sous-modulaire.*

Propriété 6. *La sous-modularité est également préservée par passage au résiduel : si $g : 2^V \to \mathbb{R}$ est sous-modulaire et $A, B \subseteq V$ sont des ensembles disjoints, alors le résiduel $f : 2^A \to \mathbb{R}$ défini par $f(S) = g(S \cup B) - g(B)$ est sous-modulaire. Les fonctions sous-modulaires croissantes le restent également par passage à la troncature si $g : 2^V \to \mathbb{R}$ est sous-modulaire, alors $f(S) = \min\{g(S), a\}$ est sous-modulaire pour toute constante a.*

La propriété 6 peut être démontrée facilement en utilisant la définition de la sous-modularité basée sur les dérivées discrètes. Cette propriété est extrêmement utile car elle permet de construire des fonctions sous-modulaires complexes à partir de fonctions plus simples (voir, par exemple, [LKG+07, SK10]). Quant à la propriété 5, elle est également simple à démontrer en utilisant la définition de la sous-modularité. Il est intéressant de constater qu'il existe de nombreux liens naturels entre les fonctions sous-modulaires et les fonctions convexes et concaves. Par exemple, pour une fonction $g : \mathbb{N} \to \mathbb{R}$, la fonction d'ensemble $f(S) = g(|S|)$ est sous-modulaire si et seulement si g est concave. Tout comme les fonctions convexes qui peuvent être minimisées efficacement, les fonctions sous-modulaires peuvent être maximisées efficacement. Il est important de souligner que les fonctions sous-modulaires peuvent également être minimisées en temps polynomial (voir par exemple [Sch00]).

Remarque 5 ([Sat05])**.** *En général, le minimum et le maximum de deux fonctions sous-modulaires ne sont pas sous-modulaires. Autrement dit, pour des fonctions sous-modulaires f_1 et f_2, les fonctions $f_{\min}(S) = \min(f_1(S), f_2(S))$ et $f_{\max}(S) = \max(f_1(S), f_2(S))$ ne sont pas nécessairement sous-modulaires.*

Dans la suite, nous considérons une autre famille de fonctions, à savoir les fonctions super-modulaires. Cette famille revêt une importance tout aussi cruciale dans le domaine de l'optimisation que la famille des fonctions sous-modulaires.

3. Une combinaison linéaire positive de fonctions positives est appelée combinaison conique.

2.2.2 Fonctions super-modulaires

Avant de présenter les fonctions super-modulaires, mentionnons la famille des fonctions modulaires. Les fonctions modulaires sont analogues aux fonctions linéaires, et elles peuvent être considérées comme le point d'intersection entre les fonctions sous-modulaires et les fonctions super-modulaires. Nous allons maintenant les définir de manière explicite.

Définition 32 (modularité). *Une fonction $f : 2^V \to \mathbb{R}$ est modulaire si pour tout $A, B \subseteq V$ et $e \notin A \cup B$, nous avons $G_f(e|A) = G_f(e|B)$. De manière équivalente, une fonction $f : 2^V \to \mathbb{R}$ est modulaire si pour tout $A, B \subseteq V$, nous avons : $f(A \cap B) + f(A \cup B) = f(A) + f(B)$.*

Un exemple classique de fonction modulaire est la fonction de cardinalité d'un ensemble $f(S) = |S|$, qui est une fonction modulaire. Il est également à noter que si une fonction f est modulaire, alors pour tout ensemble non vide $S \subseteq V$, nous avons $f(S) = f(\varnothing) + \sum_{e \in S} f(e)$.

Propriété 7. *La modularité est préservée par combinaison linéaire. Formellement, si $f_1, \ldots, f_d : 2^V \to \mathbb{R}$ sont des fonctions modulaires et $\lambda_1, \ldots, \lambda_d$ sont des constantes réelles, alors la fonction $f(S) = \sum_{i=1}^{d} \lambda_i f_i(S)$ est également modulaire.*

Dans la suite, nous allons nous concentrer sur un autre type de fonction, à savoir les fonctions super-modulaires.

Définition 33 (super-modularité [Sat05]). *Une fonction $f : 2^V \to \mathbb{R}$ est super-modulaire si pour tout $A \subseteq B \subseteq V$ et $e \in V \setminus B$, on a $G_f(e|B) \geq G_f(e|A)$. De manière équivalente, une fonction $f : 2^V \to \mathbb{R}$ est super-modulaire si pour tout $A, B \subseteq V$, nous avons : $f(A) + f(B) \leq f(A \cap B) + f(A \cup B)$.*

Propriété 8. *La fonction $f : 2^V \to \mathbb{R}$ est super-modulaire si et seulement si la fonction $g(S) = -f(S)$ est sous-modulaire.*

La propriété 8 découle naturellement de l'inversion de l'inégalité de la sous-modularité lorsque les valeurs sont multipliées par -1.

Exemple 21. *Soient $V = \{1, ..., d\}$ et $S = \{S_1, S_2, \ldots, S_d\}$ une famille d'ensembles d'éléments et soit A un sous-ensemble d'indices dans $V = \{1, ..., d\}$. Soit S_A l'intersection des S_i tels que i est un élément de A. Alors la fonction $f(A) = |S_A| = |\bigcap_{i \in A} S_i|$ est super-modulaire, positive et décroissante.*

Il est évident que la fonction $f(A)$ est positive, $f(A)$ est décroissante parce qu'à mesure d'ajouter plus d'indices à l'ensemble A, l'intersection S_A devient de plus en plus petite, ce qui signifie que $|S_A|$ diminue. $f(A)$ est super-modulaire car pour tout $A \subseteq B$ et $i \notin B$, $f(A \cup \{i\}) - f(A) = |S_A \setminus S_i|$, tandis que $f(B \cup \{i\}) - f(B) = |S_B \setminus S_i|$. Cette dernière quantité est souvent supérieure à la première, car $S_A \subseteq S_B$.

Comme la sous-modularité, la super-modularité n'est pas nécessairement préservée lorsqu'on effectue des combinaisons linéaires arbitraires, à moins que la combinaison des fonctions ne soit conique.

Proposition 4. *La super-modularité n'est pas nécessairement préservée par combinaison linéaire. Formellement, si $f_1, \ldots, f_d : 2^V \to \mathbb{R}$ sont des fonctions super-modulaires et $\lambda_1, \ldots, \lambda_d$ des constantes réelles, alors la fonction $f(S) = \sum_{i=1}^{d} \lambda_i f_i(S)$ n'est pas nécessairement super-modulaire. De plus, la somme pondérée d'une fonction super-modulaire et d'une fonction modulaire n'est pas nécessairement une fonction super-modulaire.*

Tout comme la proposition 3, pour démontrer la proposition 4, il suffit de prendre $d = 1$ et $\lambda_1 < 0$. Si nous imposons $\lambda_i \geq 0$ (pour tout i), nous obtenons que la combinaison linéaire positive de fonctions super-modulaires reste super-modulaire.

Dans la suite, et tout comme dans le cas de la sous-modularité, nous verrons qu'une combinaison linéaire positive de fonctions super-modulaires reste super-modulaire.

Lemme 1 ([CBP14]). *Soit V un ensemble fini de variables et $f : 2^V \to \mathbb{R}$ une fonction super-modulaire décroissante, et soit $a \geq 0$ une constante. Alors $g(S) = \max\{f(S), a\}$ est super-modulaire.*

Remarque 6. *En général, comme pour les fonctions sous-modulaires, le minimum et le maximum de deux fonctions super-modulaires ne sont pas super-modulaires. Autrement dit, pour des fonctions super-modulaires f_1 et f_2, les fonctions $f_{\min}(S) = \min(f_1(S), f_2(S))$ et $f_{\max}(S) = \max(f_1(S), f_2(S))$ ne sont pas nécessairement super-modulaires.*

Lemme 2 ([Sat05]). *Soit V un ensemble fini de variables, $f : 2^V \to \mathbb{R}$ et $g : 2^V \to \mathbb{R}$ deux fonctions super-modulaires décroissantes, et λ_1 et λ_2 deux constantes telles que $\lambda_1 \geq 0$ et $\lambda_2 \geq 0$. Alors $\lambda_1 f + \lambda_2 g$ est également super-modulaire décroissante.*

Remarque 7. *Nous tenons à préciser que la proposition 4 et le lemme 2 ne sont pas contradictoires. Il est important de souligner que la combinaison linéaire arbitraire de fonctions super-modulaires n'aboutit pas nécessairement à une fonction super-modulaire. Cependant, la somme de deux fonctions super-modulaires reste elle-même super-modulaire, et d'une manière générale, une combinaison conique de fonctions super-modulaires est une fonction super-modulaire.*

Après avoir étudié les propriétés des fonctions sous-modulaires et super-modulaires, il est temps d'explorer leurs applications pratiques. Ces fonctions jouent un rôle crucial dans la résolution de problèmes NP-complets. En exploitant les propriétés de sous-modularité et de super-modularité, nous pouvons formuler ces problèmes de manière à tirer parti de ces fonctions. Dans la prochaine section, nous explorons quelques exemples d'applications des fonctions sous-modulaires et super-modulaires à la résolution de problèmes NP-complets, tels que la sélection d'ensemble, l'ordonnancement. Ces exemples illustreront comment les concepts de sous-modularité et de super-modularité peuvent être utilisés de manière astucieuse pour aborder des problèmes complexes et obtenir des solutions de qualité en temps raisonnable.

2.3 Algorithmes approchés

En optimisation combinatoire, de nombreux problèmes d'optimisation sont considérés comme intraitables, ce qui signifie qu'il est peu probable de les résoudre exactement en temps polynomial. Une approche courante consiste à utiliser des méthodes d'approximation qui garantissent une qualité de solution satisfaisante en temps polynomial. Les algorithmes d'approximation avec garantie de performance sont des méthodes non exactes qui fournissent des solutions réalisables en temps polynomial tout en donnant des bornes sur leur qualité. Ces algorithmes ont été développés dès les années 1960 [Viz64, Gra66] avant la formalisation de la théorie de la NP-complétude par [COO71]. Une définition formelle des algorithmes d'approximation a été proposée par [Gar72].

Dans cette section, nous présentons quelques concepts généraux pourtant sur les algorithmes d'approximation. Pour plus de détails sur les algorithmes d'approximations, nous recommandons aux lecteurs la lecture les livres de David P. Williamson [WS11] et celui de Vijay Vazirani [Vaz13]. Ensuite, nous nous concentrons plus spécifiquement sur la maximisation de fonctions sous-modulaires et la minimisation de fonctions super-modulaires.

2.3.1 Généralités sur l'approximation

Soit une instance I d'un problème d'optimisation π. Nous notons $Sol(\pi, I)$ l'ensemble des solutions réalisables de I. Si $s \in Sol(\pi, I)$, alors $Val(I, s)$ représente la valeur associée à cette solution. $Opt(I)$ est utilisé pour désigner la valeur optimale de I, qui peut être soit un minimum, soit un maximum. De manière formelle, les solutions approchées sont définies de la manière suivante.

Définition 34 (solution approchée). *Une solution réalisable $s \in Sol(\pi, I)$ est γ-approchée pour une instance I de π avec $\gamma \in]0, 1]$ pour un problème de maximisation (ou $\gamma > 1$ pour un problème de minimisation), si sa valeur $Val(I, s)$ satisfait $Val(I, s) \geq \gamma \cdot Opt(I)$ pour un problème de maximisation (ou $Val(I, s) \leq \gamma \cdot Opt(I)$ pour un problème de minimisation). Cette condition assure que la solution est garantie d'être à un facteur ϵ de l'optimum $Opt(I)$.*

Définition 35 (algorithme d'approximation). *Un algorithme $\mathcal{A}$ est un algorithme d'approximation γ-approchée s'il retourne une solution γ-approchée pour toute instance I de π. L'algorithme $\mathcal{A}$ est polynomial s'il s'exécute en temps polynomial en fonction de la taille de l'instance d'entrée $|I|$.*

Le facteur γ, également appelé rapport d'approximation ou performance garantie, vérifie $\gamma \in [0, 1]$ pour les problèmes de maximisation et $\gamma > 1$ pour les problèmes de minimisation. L'objectif est de trouver un algorithme polynomial qui fournit le rapport d'approximation le plus proche de 1.

Définition 36 (approximabilité). *Un problème d'optimisation π est γ-approximable pour une constante γ donnée s'il existe un algorithme γ-approché pour π.*

Le rapport d'approximation γ peut être constant ou dépendre de l'instance I. Lorsque γ est constant, l'algorithme $\mathcal{A}$ est dit à rapport constant. Lorsque le rapport d'approximation n'est pas constant, il peut dépendre d'un paramètre α, où α peut prendre des valeurs dans $\mathbb{N}$, $\mathbb{R}$, $[0, 1]$, etc. L'objectif final est de fournir une garantie dans le pire des cas. Certains problèmes pour lesquels des algorithmes d'approximation à rapport constant existent peuvent bénéficier de rapports d'approximation très proches de 1. Dans ces cas, le défi consiste à trouver des algorithmes qui fournissent des rapports d'approximation aussi proches de 1 que souhaité. En général, les problèmes d'optimisation qui admettent des rapports d'approximation constants sont considérés comme les moins difficiles. En revanche, pour les problèmes qui n'admettent pas de rapport d'approximation constants, en plus de rechercher le rapport d'approximation le plus proche de 1, on peut s'intéresser aux limites de cette approximation, c'est-à-dire démontrer l'impossibilité d'atteindre certains rapports d'approximation.

Définition 37 (inapproximabilité). *Un problème de minimisation π est $(\gamma + \epsilon)$-inapproximable si pour tout $\epsilon > 0$ il n'existe aucun algorithme polynomial $(\gamma + \epsilon)$-approché pour π, à moins que* P = NP. *De manière symétrique, un problème de maximisation π est $(\gamma - \epsilon)$-inapproximable si pour tout $\epsilon > 0$ il n'existe aucun algorithme polynomial $(\gamma - \epsilon)$-approché pour π, à moins que* P = NP.

Si un problème est γ-inapproximable, il est donc peu vraisemblable qu'il existe un algorithme polynomial γ-approché. Il est alors intéressant de connaître la valeur du plus petit γ d'un problème de maximisation (ou le plus grand γ d'un problème de minimisation) pour lequel le problème est inapproximable.

Exemple 22 (inapproximabilité [Hås01]). *Un exemple de l'inapproximabilité est bien le problème MAX-3SAT qui est le problème de trouver une assignation maximisant le nombre de clauses satisfaites dans une* CNF, *dont chaque clause contient au plus 3 littéraux. Ce problème est $(8/7-\epsilon)$-inapproximable.*

Dans la suite, nous nous concentrons sur l'approximation des problèmes NP-difficiles lorsque la fonction objectif que nous cherchons à maximiser est sous-modulaire, et à minimiser quand elle est super-modulaire.

2.3.2 Maximisation sous-modulaire

Dans la suite de ce mémoire, nous utilisons les deux notations suivantes : L_f et G_f. Étant donnée une fonction d'ensemble $f : 2^{[d]} \to \mathbb{R}$, où $[d]$ est un ensemble d'entiers, les quantités suivantes capturent respectivement la perte marginale de retirer un élément i d'un ensemble S et le gain marginal d'ajouter un élément i à un ensemble S :

$$L_f(i|S) = f(S \setminus \{i\}) - f(S)$$

$$G_f(i|S) = f(S \cup \{i\}) - f(S)$$

Nous rappelons qu'une fonction f est décroissante si $L_f(i|S) \geq 0$ et elle est dite croissante si $G_f(i|S) \geq 0$ pour tous les $S \subseteq [d]$ et $i \in S$.

Maximisation des fonctions sous-modulaires. Les fonctions sous-modulaires sont couramment utilisées dans de nombreuses applications, ce qui rend l'étude de l'optimisation sous-modulaire essentielle. Il convient de souligner qu'il existe une vaste littérature dédiée à la maximisation des fonctions sous-modulaires [Sat05, Sch03]. Dans cette section, nous nous concentrons sur le problème de maximisation des fonctions sous-modulaires. Autrement dit, nous nous intéressons à la résolution de problèmes de la forme suivante :

$$\max_{S \subseteq V} f(S) \qquad \text{sous contraintes sur } S. \tag{2}$$

Un exemple courant et pertinent pour notre mémoire concerne les contraintes de cardinalité, où l'objectif est de restreindre la taille de l'ensemble S en imposant que $|S| \leq k$ pour une valeur donnée de k. Dans notre étude, nous nous intéressons spécifiquement à l'identification des k meilleurs éléments pour maximiser f. Cependant, même ce problème en apparence simple devient NP-difficile pour de nombreuses classes de fonctions sous-modulaires, telles que la couverture pondérée [Sat05] ou l'information mutuelle. Cela signifie qu'il n'existe pas d'algorithme efficace connu qui puisse résoudre ce problème en un temps polynomial, sauf si P = NP.

Il existe des algorithmes spécialisés pour maximiser les fonctions sous-modulaires [GSTT99, KNTB09]. Ces algorithmes sont faits pour trouver des solutions approchées de haute qualité, mais leur extensibilité est limitée par la difficulté intrinsèque du problème NP-difficile. Par conséquent, dans la suite de cette section, nous nous concentrons sur le développement d'algorithmes efficaces qui offrent des garanties théoriques d'approximation. Ces algorithmes visent à trouver des solutions qui se rapprochent de l'optimalité dans des délais raisonnables. Cette approche permet d'obtenir des résultats pratiques tout en tenant compte de la complexité du problème.

Algorithme glouton. Nous avons rappelé que le problème de maximisation d'une fonction sous-modulaire (croissante) sous contrainte de cardinalité est NP-difficile. Une approche simple pour résoudre le problème 2 dans le cas des contraintes de cardinalité est l'algorithme glouton proposé par [NWF78]. Cet algorithme commence avec un ensemble vide S_0 et à chaque itération j, il ajoute un élément qui maximise le gain $G_f(e|S_{j-1})$. L'algorithme est détaillé ci-après (algorithme 2).

Un résultat célèbre de [NWF78] démontre que l'algorithme glouton 2 fournit une bonne approximation de la solution optimale du problème d'optimisation considéré.

Théorème 2 (Nemhauser et al. 1978). *Soit une fonction sous-modulaire et positive $f : 2^{[d]} \to \mathbb{R}^+$, et soient $\{S_j\}_{j \geq 0}$ les ensembles sélectionnés de manière gloutonne par l'algorithme 2. Alors, pour tous les*

Algorithm 2 Algorithme glouton pour la maximisation d'une fonction sous-modulaire croissante

Input: une fonction f sous-modulaire croissante, un ensemble $X \subseteq [d]$, un entier $k \leq d$.
Output: un sous-ensemble S_k tel que $|S_k| \leq k$.

$S_0 \leftarrow \varnothing$
for $j \in \{1, \cdots, k-1\}$ **do**
  $e^* \longleftarrow \underset{e \in X}{\operatorname{argmax}} \{G_f(e|S_j)\}$
  $S_{j+1} \longleftarrow S_j \cup \{e^*\}$
  $X \leftarrow X \setminus \{e^*\}$
return S_k

entiers positifs l et k, nous avons : $f(S_l) \geq (1 - e^{-\frac{l}{k}}) \max_{S:|S|\leq k} f(S)$ pour tous les ensembles S tels que $|S| \leq k$. En particulier, lorsque $l = k$, $f(S_k) \geq (1 - \frac{1}{e}) \max_{|S|\leq k} f(S)$.

La possibilité de généraliser légèrement en permettant que $k \neq l$ présente de nombreux avantages. Par exemple, si nous autorisons l'algorithme glouton 2 à sélectionner $5k$ éléments, le rapport d'approximation par rapport à l'ensemble optimal de taille k s'améliore significativement, passant d'environ 0.63 à près de 0.99 [Von07].

Remarque 8. *Pour la classe de fonctions sous-modulaires, le résultat précédent est le meilleur qui puisse être obtenu avec n'importe quel algorithme glouton efficace.*

En effet, [NWF78] ont prouvé que tout algorithme qui se limite à évaluer f sur un nombre polynomial d'ensembles ne peut pas garantir une approximation meilleure que $1 - e^{-1}$.

2.3.3 Minimisation super-modulaire

Nous rappelons tout d'abord que, contrairement à la maximisation de fonctions sous-modulaires, la minimisation de fonctions super-modulaires est généralement difficile [FKT15]. Cette difficulté découle du fait qu'une valeur nulle de la fonction objectif peut conduire un algorithme d'approximation à trouver une solution optimale, et que la maximisation de la négation d'une fonction super-modulaire avec une constante est un problème difficile.

Minimisation super-modulaire. La minimisation de fonctions super-modulaires est importante car elle permet de modéliser divers problèmes pratiques, de trouver des solutions efficaces sous contraintes de cardinalité, d'utiliser des approximations et des heuristiques performantes, et de contribuer à la compréhension théorique de l'optimisation. Il est à noter que les fonctions sous-modulaires sont couramment utilisées dans de nombreuses applications, ce qui rend aussi l'étude de l'optimisation super-modulaire aussi essentielle que l'optimisation sous-modulaire. Il existe également une vaste littérature dédiée à la minimisation des fonctions super-modulaires [FKT15, CBP14, CABP14, Il'01]. Dans ce travail, nous nous intéressons à la résolution de problèmes de la forme suivante :

$$\min_{S \subseteq V} f(S) \qquad \text{sous contraintes sur } S. \tag{3}$$

Tout comme la maximisation sous-modulaire, nous nous intéressons aux contraintes de cardinalité, où l'objectif est de restreindre la taille de l'ensemble S en imposant $|S| \leq k$ pour une valeur donnée de k. Dans notre étude, tout comme dans le cas de la maximisation sous-modulaire, nous nous intéressons spécifiquement à l'identification du meilleur ensemble d'une taille au plus k qui minimise $f \geq 0$. La

minimisation super-modulaire est généralement bien plus difficile que la maximisation sous-modulaire. Pour la minimisation d'une fonction super-modulaire sous contrainte de cardinalité, il est important de préciser que le problème est non seulement NP-difficile, mais aussi qu'il n'est pas approchable à une constante près [MS13]. Néanmoins, il existe des algorithmes spécialisés pour minimiser les fonctions sous-modulaires [CBP14, Il'01, LS17]. Ces algorithmes sont faits pour trouver des solutions approchées de haute qualité. Dans la suite de cette section, nous nous concentrons sur le développement d'algorithmes efficaces qui offrent des garanties théoriques d'approximation.

Algorithme glouton. Le problème de minimisation d'une fonction super-modulaire (décroissante) sous contrainte de cardinalité est difficile. Une approche simple pour résoudre le problème 3 dans le cas des contraintes de cardinalité est l'algorithme glouton proposé par [LS17]. L'idée générale de cet algorithme est de produire des approximations précises en partant de l'ensemble vide et en augmentant progressivement la limite de taille des solutions. L'algorithme est détaillé ci-dessous :

Algorithm 3 Algorithme glouton pour la minimisation d'une fonction super-modulaire décroissante

Input: fonction super-modulaire et positive $f(S)$, $\alpha > 0$, un ensemble $X \subseteq [d]$, un entier $k \leq d$.
Output: un sous-ensemble S_k.

$S_0 \leftarrow \varnothing$
$j \leftarrow 0$
while $j \leq \left\lceil k \ln\left(\frac{f(S_0)}{\alpha f(S_{j-1})}\right)\right\rceil$ **do**
  $e^* \longleftarrow \underset{e \in X}{\operatorname{argmin}} \{f(S \cup \{e\}\}$
  $S_{j+1} \longleftarrow S_j \cup \{e^*\}$
  $X \leftarrow X \setminus \{e^*\}$
  $j \leftarrow j + 1$
$S_k \leftarrow S_{j+1}$
return S_k

Lemme 3 ([LS17])**.** *Soit f une fonction super-modulaire non négative et S_τ la sortie de l'algorithme 3 pour une erreur $\alpha > 0$. Alors $|S_\tau| \leq |S_0| + \left\lceil \alpha k \ln\left(\frac{f(S_0)}{\alpha f(S^*)}\right)\right\rceil$ et $f(S_\tau) \leq \frac{f(S^*)}{1-\alpha}$, où S^* est la solution optimale de 3 avec des contraintes de cardinalité $|S| \leq k$.*

Remarque 9 ([MS13])**.** *Pour la minimisation d'une fonction super-modulaire sous contrainte de cardinalité, il est important de préciser que le problème est non seulement NP-difficile, mais aussi qu'il n'est pas approchable à une constante près. Ce qui justifie l'utilisation de la notion de* ***courbure****.*

2.3.4 Importance de la courbure

Nous avons indiqué que le meilleur rapport d'approximation constant possible à obtenir pour les fonctions sous-modulaires est $(1 - e^{-1})$[NWF78]. Ce résultat est le meilleur qui puisse être obtenu avec n'importe quel algorithme efficace pour la maximisation sous-modulaire. Par la suite, [Von10] a fourni des résultats plus affinés sur le meilleur facteur d'approximation possible en fonction de la courbure de f. La même conclusion peut être tirée pour la minimisation super-modulaire.

Définition 38 (**courbure**)**.** *Pour une fonction d'ensemble f sur 2^V et $I \subseteq V$, la courbure de f sur l'ensemble 2^I est définie comme suit :*

$$c = 1 - \min_{i \in I} \left[\frac{L_f(i|I)}{L_f(i|\{i\})}\right] = 1 - \min_{i \in I} \left[\frac{G_f(i|I \setminus \{i\})}{G_f(i|\{\varnothing\})}\right]$$

Proposition 5 (**courbure** [NWF78]). *Soit une fonction d'ensemble $f : 2^{[d]} \to [0,1]$, et $I \subseteq [d]$ un ensemble non vide. Si f est positive, super-modulaire et décroissante ou sous-modulaire et croissante et positive, alors la courbure c de f sur 2^I satisfait $0 \leq c \leq 1$. Nous précisons que toute fonction modulaire a une courbure de 0 (plus la courbe s'approche de 0, plus on est modulaire).*

Exemple 23 (**courbure**). *Considérons un ensemble fini $V = \{1,2,3\}$ et la fonction sous-modulaire $f(S) = \sqrt{|S|}$ où $|S|$ est le cardinal de l'ensemble S, la courbure totale de f sur 2^V est donnée par :*

$$c = 1 - \min_{i \in V} \frac{G_f(i|V \setminus \{i\})}{G_f(i|\{\varnothing\})} = 1 - (\sqrt{3} - \sqrt{2}) \approx 0.682$$

Il est intéressant de noter que la courbure est liée à la notion de « *steepness* » telle que définie dans [Il'01]. Lorsque $I = [d]$, nous parlons de la courbure totale de f [CC84]. Une observation remarquable est que, dans le cas où f est super-modulaire et décroissante, la condition $0 < c < 1$ est suffisante pour garantir qu'il est possible d'approcher la tâche de minimisation de f sous une contrainte de cardinalité à une constante près [Il'01, LS17].

Sous-modularité et courbure. Soient (E, M) un matroïde et $f : 2^E \to \mathbb{R}^+$ une fonction sous-modulaire croissante. La courbure de f est un paramètre $c \in [0,1]$ tel que pour tout $S \subset E$ et $j \in E \setminus S$, $f(S \cup \{j\}) - f(S) \geq (1-c) f(\{j\})$. Maintenant, nous considérons le problème d'optimisation 2 sous contraintes de cardinalité sur un matroïde. Il est connu que l'algorithme glouton donne un rapport d'approximation constant de $\frac{1}{2}$ pour ce problème [NW78], et un rapport d'approximation non constant de $\frac{1}{1+c}$ lorsque f a une courbure c [CC84]. Pour le matroïde uniforme, il est connu que l'algorithme glouton donne une approximation améliorée de $\frac{1}{c}(1 - e^{-c})$ [CC84].

Les auteurs de l'article [Von08] ont analysé la version continue de l'algorithme glouton 2 et ont prouvé qu'il donne une approximation de $\frac{1}{c}(1 - e^{-c})$ pour le problème 2 sous contraintes de cardinalité sur un matroïde, et ce, pour n'importe quel matroïde. D'autre part, les auteurs de [Von10] ont montré qu'avec une courbure c par rapport à l'optimum, il est impossible d'obtenir une meilleure approximation que $\frac{1}{c}(1 - e^{-c})$ en utilisant un nombre polynomial de requêtes de valeur.

Super-modularité et courbure. Comme nous l'avons indiqué précédemment, le problème de minimisation d'une fonction sous-modulaire décroissante f sur un ensemble fini de variables V et soumise à une contrainte de cardinalité est difficile à approcher en général. Ainsi, pour obtenir un résultat d'approximation, diverses hypothèses sur la fonction objectif f doivent être faites. Comme suggéré dans [Il'01, LS17], l'une de ces hypothèses que nous allons exploiter dans notre étude s'appuie sur la notion de courbure c. Une condition nécessaire pour garantir l'approximabilité du problème de minimisation de la fonction f sous contrainte de cardinalité liée directement à la courbure c de cette fonction sur 2^V est $c < 1$.

Exemple 24 (**courbure d'une fonction modulaire**). *Considérons un ensemble fini $V = \{1,2,3\}$ et la fonction modulaire $f(S) = |S|$ où $|S|$ est le cardinal de l'ensemble S, la courbure totale de f sur 2^V est donnée par :*

$$c = 1 - \min_{i \in V} \left[\frac{G_f(i|V \setminus \{i\})}{G_f(i|\{\varnothing\})} \right] = 1 - 1 = 0$$

2.4 Conclusion

Dans ce chapitre, nous avons présenté les notions de base et les concepts d'optimisation combinatoire nécessaires à la compréhension du chapitre 8 de cette thèse. Nous avons brièvement rappelé les notions de matroïdes et de programmation linéaire, qui sont des concepts fondamentaux de l'optimisation combinatoire. Ensuite, nous avons examiné plus en détail les fonctions sous-modulaires, les fonctions modulaires, les fonctions super-modulaires et les différentes propriétés associées à ces trois familles de fonctions.

Nous avons également abordé le problème de la maximisation sous-modulaire et de la minimisation super-modulaire sous contraintes de cardinalité. Nous avons constaté que ces deux problèmes sont difficiles. Nous avons présenté différents algorithmes gloutons de l'état de l'art permettant de calculer des solutions approchées avec des garanties mathématiques, en mettant l'accent sur divers résultats théoriques bien connus. Enfin, nous avons souligné l'importance de la courbure et indiqué comment celle-ci peut être exploitée pour améliorer les résultats d'approximation.

3

Apprentissage automatique

Sommaire

3.1 Apprentissage automatique et classification 43
3.1.1 Types d'apprentissage 44
3.1.2 Apprentissage supervisé et classification 44
3.1.3 Modèles d'apprentissage automatique 46
3.2 Modèles symboliques (basés sur des règles) 47
3.2.1 Arbres de décision (*decision trees*) 48
3.2.2 Listes de décisions (*decision lists*) 50
3.2.3 Règles de décisions (*decision rules*) 50
3.3 Modèles ensemblistes 52
3.3.1 Forêts aléatoires 53
3.3.2 Boosting d'arbres 54
3.3.3 Quelques statistiques sur les arbres de décision et les ensembles d'arbres . 55
3.4 Conclusion 58

Ce chapitre a pour objectif de présenter quelques notions élémentaires en apprentissage automatique, qui sont utiles dans les prochains chapitres. Nous débutons par une introduction à l'apprentissage statistique supervisé, en nous focalisant essentiellement sur la classification. en abordant les différents types d'attributs (*features*), ainsi que les principaux types d'apprentissage. Ensuite, nous définissons la classification supervisée. Enfin, nous présentons des modèles d'apprentissage automatique reconnus pour leur interprétabilité, tels que les modèles basés sur des règles (dits symboliques). Nous concluons ce chapitre en abordant les modèles d'apprentissage ensemblistes et en explorant différents concepts qui leur sont liés.

3.1 Apprentissage automatique et classification

L'apprentissage automatique (*machine learning*), est une discipline de l'intelligence artificielle qui permet d'apprendre à partir de données. Il s'agit d'une approche puissante pour résoudre des problèmes complexes en exploitant des modèles et des structures présents dans les données. Le paradigme de l'apprentissage statistique (supervisé) consiste à extrapoler à partir d'un échantillon de données étiquetées un modèle qui permet de prédire l'étiquette de nouvelles données. Le processus d'apprentissage consiste à découvrir des relations, des motifs ou des tendances cachés dans les données, ce qui permet au modèle d'effectuer des généralisations et de faire des prédictions sur des données non vues auparavant. L'apprentissage automatique est largement utilisé dans de nombreux domaines, tels que la reconnaissance

vocale, la vision par ordinateur, la recommandation de produits, la détection de fraude, l'analyse de sentiments, la prédiction des tendances, etc. L'éventail des modèles utilisés en apprentissage automatique est très large, s'étendant depuis les arbres de décision et règles de décision jusqu'aux méthodes ensemblistes, aux modèles à noyaux et aux réseaux de neurones. En résumé, l'apprentissage automatique offre une approche puissante pour automatiser des tâches complexes. Il est au cœur de nombreuses avancées technologiques.

3.1.1 Types d'apprentissage

Globalement, un modèle d'apprentissage automatique est une correspondance qui associe des entrées à des prédictions. Un jeu de données (*benchmark* ou *dataset*) est généralement un ensemble de vecteurs d'attributs dans le cas tabulaire. Le nombre d'attributs d'un jeu de données est appelé dimension. Un vecteur d'attributs est également appelé instance. Le processus de génération de modèles à partir de jeux de données est appelé apprentissage ou entraînement. Pour explorer davantage d'exemples et avoir une revue historique concis, nous recommandons aux lecteurs le livre « *The Elements of Statistical Learning* » de J.H. Friedman [HTF01].

Nous nous concentrerons principalement sur *l'apprentissage supervisé*, qui constitue le cœur de notre étude. L'apprentissage supervisé implique l'entraînement de modèles à partir de données étiquetées, où des caractéristiques d'entrée (les valeurs des attributs appelés variables explicatives) sont associés à des étiquettes correspondantes. Le modèle apprend à associer les caractéristiques aux étiquettes en identifiant des schémas et des relations dans les données, leur permettant de prédire avec précision les étiquettes des nouvelles données non étiquetées. Il convient de noter qu'il existe d'autres types d'apprentissage, tels que *l'apprentissage non supervisé*, qui se concentre sur la découverte de structures internes dans des données non étiquetées, ainsi que *l'apprentissage par renforcement*, qui permet à un agent d'apprendre à prendre des décisions en interagissant avec un environnement pour maximiser une récompense. Bien que ces approches soient essentielles dans divers contextes, notre attention se portera principalement sur l'apprentissage supervisé, qui constitue l'objet de notre recherche.

Les modèles d'apprentissage automatique reposent sur les données, qui sont généralement présentées sous forme d'un ensemble d'attributs, pouvant être de différentes natures. Dans la suite, nous allons présenter quelques types d'attributs couramment utilisés dans l'apprentissage automatique :

- **attributs numériques :** ce sont des attributs qui représentent des valeurs numériques réelles continues, telles que le revenu mensuel.
- **attributs catégoriels :** ce sont des attributs qui représentent des valeurs de catégories ou des classes discrètes, telles que le genre d'une personne (homme/femme) ou la couleur d'une voiture (noire, jaune, rouge, bleue, etc.).
- **attributs textuels :** ce sont des attributs qui représentent du texte brut, comme des tweets ou des commentaires clients sur un produit commercial.
- **attributs images :** ce sont des attributs qui représentent des images brutes, par exemple des images de chats ou de chiens.

3.1.2 Apprentissage supervisé et classification

Il existe diverses tâches d'apprentissage supervisé qui peuvent se décliner selon le choix de l'ensemble des sorties. Lorsque $\mathcal{Y}$ est un ensemble fini de valeurs discrètes, nous parlons de *classification* et lorsque $\mathcal{Y}$ est un ensemble infini (mais compact) de valeurs continues, nous parlons de *régression*. En classification binaire, $\mathcal{Y}$ contient deux valeurs discrètes possibles, souvent notées 0 et 1, ou -1 et $+1$. En classification multi-classes, $\mathcal{Y}$ est un ensemble de la forme $\{1, \cdots, k\}$ et l'objectif est d'identifier, pour

chaque instance, une étiquette parmi les k étiquettes possibles. [4]

Une description formelle de l'apprentissage supervisé est la suivante. Nous notons $\mathcal{X}$ comme l'espace des instances (entrées) et $\mathcal{Y}$ l'espace des étiquettes (sorties). Nous rappelons que l'objectif de l'apprentissage supervisé est de déterminer une relation entre les attributs $\mathcal{X}$ (ou variables explicatives) et les étiquettes $\mathcal{Y}$, en utilisant un modèle $\mathcal{C}$ d'apprentissage automatique. Nous notons $\mathcal{C} : \mathcal{X} \rightarrow \mathcal{Y}$, où $\hat{y_i} = \mathcal{C}(x_i)$ est la prédiction faite pour l'instance x_i. Lorsque l'ensemble des étiquettes $\mathcal{Y}$ consiste en un ensemble fini de valeurs discrètes, cela correspond au problème de classification supervisée. En revanche, lorsque $\mathcal{Y}$ est constitué d'un ensemble de valeurs continues, on parle de problème de régression. Pour la classification binaire, nous utilisons généralement $\mathcal{Y} = \{0, 1\}$. Lorsque $|\mathcal{Y}| > 2$, il s'agit de classification multi-classe. Dans cette thèse, nous nous concentrons principalement sur les problèmes de classification, en particulier sur le problème de la classification binaire. Nous dérivons maintenant certaines mesures d'évaluation liées à la classification.

Mesures de performance. Considérons un problème de classification et un classifieur $\mathcal{C} : \mathcal{X} \rightarrow \mathcal{Y}$. Après que le classifieur $\mathcal{C}$ a été appris à partir d'un ensemble de données d'entraînement étiquetées $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_d, y_d)\}$, on peut utiliser $\mathcal{C}$ pour prédire l'étiquette d'une instance qui ne figure pas dans $\mathcal{D}$. La prédiction réalisée peut être correcte ou pas. Dans ce contexte : L'erreur empirique, notée $\hat{R}(\mathcal{C})$, est une mesure de la performance du modèle $\mathcal{C}$, elle est définie comme la moyenne des erreurs de prédiction sur l'ensemble $\mathcal{D}$. Formellement :

$$\hat{R}(\mathcal{C}) = \frac{1}{d} \sum_{i=1}^{d} L(\mathcal{C}(x_i), y_i)$$

où L est une fonction de perte qui mesure la différence entre la prédiction de $\mathcal{C}(x_i)$ et l'étiquette y_i. L'erreur empirique est une estimation de la capacité du modèle à bien s'ajuster aux données d'entraînement, mais elle ne prend pas en compte la capacité du modèle à généraliser sur de nouvelles données. Dans le même contexte, la précision (*accuracy*) correspond au pourcentage d'instances correctement prédites dans l'ensemble de données. Formellement, la performance de $\mathbf{C}$ sur un ensemble $\mathcal{D}'$ est donnée par :

$$\text{Accuracy}(\mathcal{C}, \mathcal{D}') = \frac{1}{|\mathcal{D}'|} \sum_{i=1}^{|\mathcal{D}'|} 1_{\{\hat{y_i'} = y_i'\}}$$

où $\hat{y_i'} = \mathcal{C}(x_i')$ est la prédiction faite pour l'instance x_i' par le classifieur $\mathcal{C}$, et $1_{\hat{y_i'} = y_i'}$ est une fonction indicatrice. Cette fonction renvoie 1 si la valeur prédite $\hat{y_i'}$ est égale à la vraie valeur y_i', et elle renvoie 0 si elles sont différentes. $|\mathcal{D}'|$ est le nombre d'instances dans l'ensemble de test. Nous avons, $\text{Accuracy}(\mathcal{C}, \mathcal{D}') \in [0, 1]$. Nous définissons l'erreur de classification comme : Error = 1 - Accuracy.

Sur-apprentissage (*overfitting*). Nous avons toujours l'espoir que le classifieur $\mathcal{C}$ présente de bonnes performances de prédiction. Cependant, comme nous n'avons accès qu'aux données d'entraînement, seule l'erreur empirique est accessible. Malheureusement, il est fréquent qu'un classifieur ayant une faible erreur empirique sur l'ensemble d'apprentissage considéré se comporte mal avec les instances non vues de l'ensemble de test utilisé pour évaluer la précision. Ce phénomène est appelé sur-apprentissage (*overfitting*). L'une des principales raisons du sur-apprentissage est que le classifieur est si performant sur l'ensemble d'entraînement qu'il considère certaines propriétés spécifiques de cet ensemble comme étant

4. Notons pour certains problèmes de classification, comme le classement d'étiquettes (*label ranking*), $\mathcal{Y}$ peut être de taille exponentielle.

des propriétés générales. Tandis que le phénomène du sous-apprentissage (*underfitting*) veut simplement dire que le classifieur se comporte mal sur les données d'entraînement, ce qui indique que le classifieur n'apprend pas suffisamment de propriétés générales à partir de l'ensemble d'entraînement. Si la dimension de Vapnik-Chervonenkis « **VC dimension (VC-dim)** » du classifieur est finie, alors avec suffisamment d'exemples, le risque empirique converge vers le vrai risque (erreur sur l'ensemble de test), ce résultat vient directement du théorème fondamental de l'apprentissage statistique [SSBD14] (théorème 6.7).

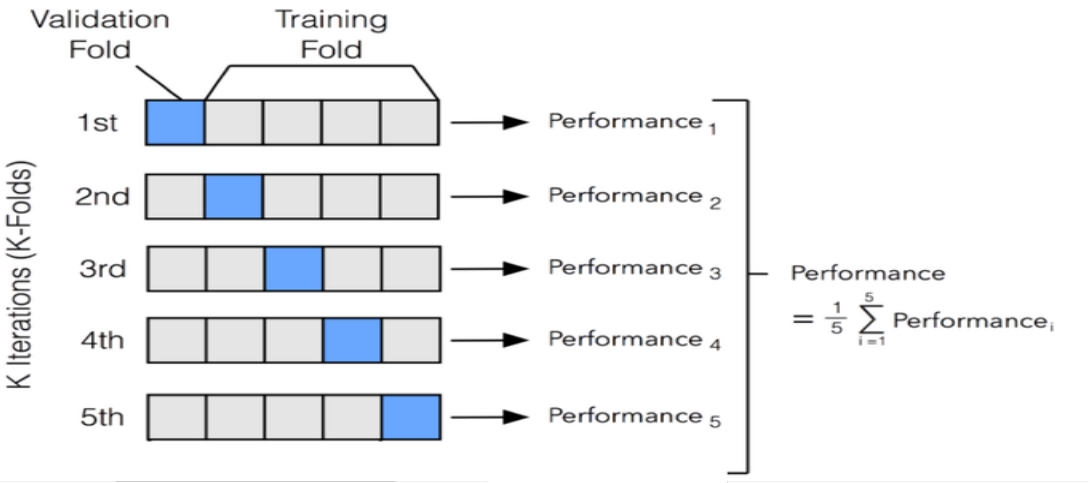


FIGURE 6 – 5-fold cross validation (source : [PVG+11]).

Validation croisée (*k-fold cross validation*). La méthode de validation croisée à k-blocs [Koh95] commence par diviser l'ensemble de données étiquetées $\mathcal{D}$ en k sous-ensembles exclusifs de tailles similaires (ou identiques). Ainsi, $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2 \cup \ldots \cup \mathcal{D}_k$, où pour $i \neq j$, on a $\mathcal{D}_i \cap \mathcal{D}_j = \varnothing$. Pour chaque sous-ensemble $\mathcal{D}_i$, les $k-1$ sous-ensembles restants sont utilisés comme ensemble d'entraînement, tandis que $\mathcal{D}_i$ est utilisé comme ensemble de test. Ensuite, k blocs d'ensembles d'entraînement-test sont générés pour l'évaluation. L'erreur d'entraînement (ou de test) finale est la valeur moyenne des erreurs d'entraînement (ou de test) sur les k blocs. La méthode de validation croisée dépend fortement de la valeur de k. Celle-ci est généralement choisie comme 5 ou 10, en prenant en compte l'équilibre entre le temps de calcul et la qualité de l'évaluation. De plus, pour rendre l'évaluation plus robuste, il arrive parfois que la validation croisée soit répétée plusieurs fois avec différentes graines aléatoires afin d'obtenir des valeurs moyennes. La figure 6 illustre le concept de la validation croisée pour $k = 5$. La méthode de validation croisée est l'une des méthodes de validation les plus populaires. Dans cette thèse, la validation croisée à k-blocs est largement utilisée dans nos expériences.

3.1.3 Modèles d'apprentissage automatique

Au cours de la dernière décennie, grâce aux avancées considérables en matière de ressources informatiques, de nombreux algorithmes d'apprentissage automatique efficaces et précis ont été développés. Notamment, les travaux pionniers de LeCun [LB95, LDH+90] sur les premiers modèles de CNN et le célèbre jeu de données ImageNet ont ouvert la voie à une série d'autres travaux importants dans le domaine des réseaux neuronaux. Parmi ceux-ci, nous pouvons citer les GANs [GPAM+14], ainsi que des modèles récents et influents tels que LLM [GG22] et BERT [DCLT19]. En plus des réseaux de neurones,

les méthodes à base d'ensembles sont également populaires. Parmi eux, on retrouve des algorithmes tels que XGBoost [CG16], LightGBM [KMF+17], les forêts aléatoires [Bre01], et le Deep Forest [ZF17]. Ces algorithmes ont rencontré un grand succès dans divers domaines d'application de l'apprentissage automatique, tels que la vision par ordinateur, le traitement du langage naturel, le traitement vidéo, la détection de fraude, etc. Cependant, les modèles appris à l'aide de ces algorithmes sont souvent considérés comme des « boîtes noires » en raison de leur manque d'interprétabilité et de la difficulté à expliquer les décisions qu'ils prennent. Bien qu'il existe différentes définitions de l'interprétabilité [Mil19, DK17], l'idée principale est que l'amélioration de l'interprétabilité permet aux êtres humains de comprendre les raisons des décisions prises ou des prédictions faites formulées par ces modèles. D'autres modèles de réseaux de neurones peuvent être moins complexes à expliquer en raison de leur architecture binaire. Parmi ces modèles, nous citons les perceptrons multicouches booléens et les réseaux de neurones binaires.

Perceptrons multicouches booléens (MLP). Les perceptrons multicouches booléens également connu sous le nom de réseaux booléens à seuil multicouches [Ant01], sont des réseaux de neurones artificiels qui sont utilisés pour des tâches de classification binaire. Ils sont composés de plusieurs couches de neurones, une couche d'entrée, une ou plusieurs couches cachées et une couche de sortie. Chaque neurone dans les couches cachées et de sortie utilise une fonction d'activation qui convertit la somme pondérée de ses entrées en une valeur de sortie binaire (0 ou 1).

Réseaux de neurones binaires (BNN). Introduits dans [HCS+16], les réseaux de neurones binaires sont des réseaux de neurones multicouches dont les activations et les poids sont principalement binaires (mais variant dans $\{-1, 1\}$). Un BNN est généralement décrit en termes de composition de d blocs de couches (assemblés séquentiellement) plutôt que de couches individuelles. Ainsi, un BNN est composé d'un certain nombre (disons $m = d - 1$) de blocs internes, suivis d'un bloc de sortie unique, noté O. Chaque bloc est constitué d'une collection de transformations linéaires et non linéaires.

Ces dernières années, de nombreuses recherches ont été conduites dans le domaine de l'apprentissage automatique afin d'améliorer l'interprétabilité des modèles. Ces travaux peuvent être divisés en deux catégories principales [Mol20] : l'interprétabilité intrinsèque et l'interprétabilité post hoc. L'interprétabilité intrinsèque concerne les modèles d'apprentissage automatique considérés comme interprétables en raison de leur structure simple, tels que les arbres de décision. En revanche, l'interprétabilité post-hoc consiste à appliquer des méthodes d'interprétation après l'apprentissage des modèles « boîte noire ». Cela implique de créer un deuxième modèle qui vise à expliquer le modèle « boîte noire » initialement entraîné. Par exemple, il est possible d'extraire un arbre de décision pour expliquer les prédictions d'un réseau neuronal [FH17]. Dans cette thèse, nous nous concentrons principalement sur les modèles d'apprentissage automatique basés sur des règles (modèles symboliques) et les modèles ensemblistes pour la classification binaire. Nous présentons plusieurs modèles largement utilisés, tels que les arbres de décision, les listes de décisions et les règles de décisions, ainsi que d'autres modèles populaires comme les modèles ensemblistes (forêts aléatoires et arbres optimisés).

3.2 Modèles symboliques (basés sur des règles)

Les modèles symboliques, ou les modèles basés sur des règles, forment une approche d'apprentissage automatique qui repose sur la représentation explicite de règles logiques. Ces modèles utilisent des ensembles de règles pour décrire les relations entre les variables et effectuer des prédictions. Ils sont souvent utilisés pour leur interprétabilité, car les règles logiques peuvent être comprises et interprétées par les utilisateurs. Les modèles symboliques sont largement utilisés dans des domaines sensibles tels

que la médecine [PKSR02] et la finance [RW17], où l'interprétabilité est essentielle. En médecine, ils aident à la classification des maladies, l'analyse génomique et la prédiction des traitements. Dans la finance, ils détectent les fraudes, évaluent les risques et prédisent les fluctuations des marchés. Ces modèles sont également utilisés dans l'exploration de données et la fouille de données pour découvrir des relations complexes, identifier des schémas et réaliser des prédictions. Ils offrent ainsi des performances prédictives et une interprétabilité précieuse dans l'analyse de données sensibles.

3.2.1 Arbres de décision (*decision trees*)

Les arbres de décision (DT) [Qui86] constituent l'un des modèles d'apprentissage automatique les plus populaires en apprentissage supervisé. Le processus de décision d'un arbre de décision est similaire à celui de l'être humain, prenant des décisions en suivant une série de règles (conjonction de règles). Cette logique simple dans le processus de décision rend l'arbre de décision interprétable. En général, un arbre de décision est composé d'un nœud racine (*root node*), de plusieurs nœuds de décision (*decision nodes*) et de nœuds feuilles (*leaf nodes*). Chaque nœud de décision de utilise un attribut comme critère de test pour évaluer une instance. Les nœuds feuilles correspondent à des classes dans le cas d'un problème de classification ou à des valeurs réelles pour un problème de régression.

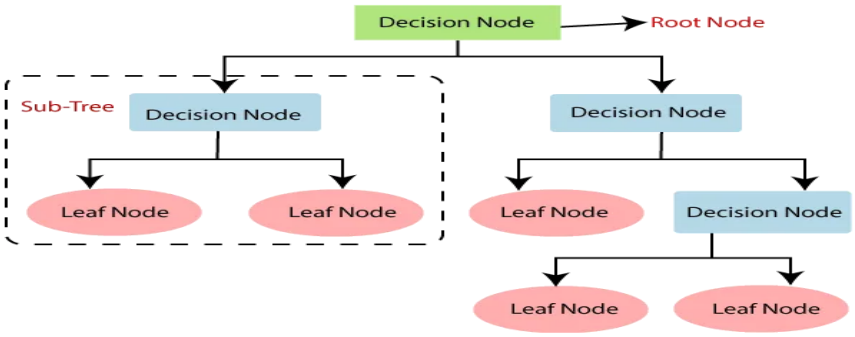


FIGURE 7 – Une structure générale d'arbre de décision (source : DT)

Un arbre de décision est présenté graphiquement comme à la figure 7. Chaque sous-arbre représente un cas correspondant à un sous-ensemble de valeurs spécifiques de l'attribut sélectionné, selon un critère bien défini. Chaque chemin, de la racine à un nœud feuille, correspond à un terme qui regroupe les conditions rencontrées dans les noeuds de décision par lesquels le chemin passe. Pour prédire la sortie d'une nouvelle donnée inconnue, il faut trouver le chemin unique compatible avec cette donnée en parcourant l'arbre à partir du nœud racine jusqu'à un nœud feuille. La prédiction finale est la valeur associée au nœud feuille. Nous allons maintenant donner une définition formelle des arbres de décision, qui sera utilisé tout au long de cette thèse.

Définition 39 (arbre de décision). *Un arbre de décision booléen sur X_d est un arbre binaire, où chaque nœud interne est étiqueté par l'une des d variables booléennes d'entrée, et pour lequel chacune des feuilles est étiquetée par 0 ou 1. Chaque variable apparaît au plus une fois sur n'importe quel chemin de la racine à feuille (read-once property). La valeur $T(\boldsymbol{x}) \in \{0,1\}$ de T pour l'instance d'entrée $\boldsymbol{x}$ est donnée par l'étiquette de la feuille atteinte depuis la racine comme suit : à chaque nœud, on suit le fils gauche ou droit selon que la valeur d'entrée de la variable correspondante est 0 ou 1. La taille de T, notée $|T|$ est donnée par le nombre de ses nœuds. La classe des arbres de décision sur X_d est notée $\mathtt{DT}_d$.*

Un exemple illustrant le concept d'arbre de décision pour la classification est présenté à la figure 8. Cet arbre a été appris à partir d'un petit ensemble de données contenant des informations standards sur des clients. L'arbre de décision de la figure 8 classe les *prêts bancaires* en utilisant les attributs suivants :

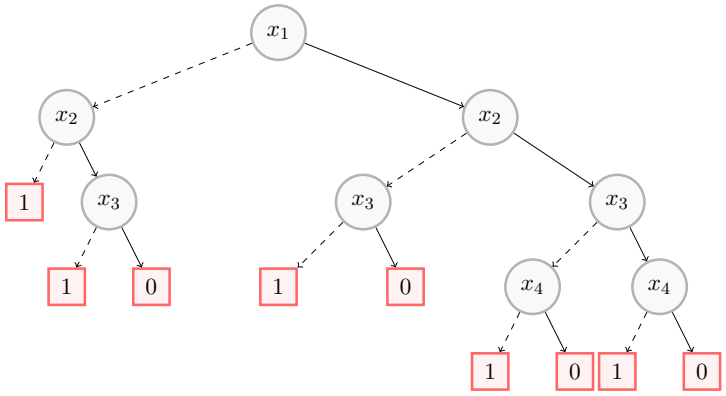


FIGURE 8 – Un arbre de décision T pour classer l'attribution d'un *prêt bancaire*

— x_1 (situation professionnelle) : « le client n'est pas en contrat CDI »
— x_2 (âge) : « le client a plus de 50 ans »
— x_3 (niveau de revenu) : « les revenus annuels du client sont inférieurs à 35K € »
— x_4 (fiabilité du client) : « le client n'a pas remboursé un prêt précédent »

Maintenant, donnons des informations sur un client traduites par l'instance $\boldsymbol{x} = (1, 1, 1, 1)$. L'arbre de la figure 8 classe cette instance comme étant une instance négative, car le chemin correspondant, en parcourant l'arbre du nœud racine jusqu'au nœud feuille, conduit à la valeur 0. Cela signifie que le prêt bancaire a été refusé.

Maintenant, nous rappelons que dans le contexte des arbres de décision, un arbre « optimal » est celui qui minimise l'erreur empirique. En d'autres termes, parmi tous les arbres de décision possibles que nous pouvons construire, l'arbre « optimal » serait celui qui minimise le plus l'erreur empirique, c'est-à-dire celui qui s'ajuste le mieux aux données d'entraînement.

Apprendre un arbre de décision. Nous rappelons tout d'abord que la complexité de l'apprentissage d'un arbre de décision optimal [NIPMS18] dépend de plusieurs facteurs, tels que la taille de l'ensemble de données d'entraînement et le nombre d'attributs. En général, l'apprentissage d'un arbre de décision optimal est un problème NP-difficile. Une méthode classique pour l'apprentissage des arbres de décision est l'utilisation d'un algorithme glouton. Chaque nœud représente un sous-ensemble de l'ensemble de données, le nœud racine correspondant à l'ensemble de données complet. Pour les nœuds de division (y compris le nœud racine), le sous-ensemble correspondant est divisé en plusieurs parties pour former des sous-arbres distincts en fonction des attributs associées. La division se termine aux nœuds feuilles lorsque tous les exemples du sous-ensemble ont la même classe (d'autres critères de décomposition peuvent être utilisés).

Dans la pratique, des algorithmes gloutons d'apprentissage d'arbres de décision tels que l'algorithme **ID3, C4.5 et CART** [PVG+11, Qui86] utilisent des heuristiques pour la construction d'un arbre de décision de manière efficace. La complexité de ces algorithmes dépend de la taille de l'ensemble de données et du nombre d'attributs. En règle générale, elle est de l'ordre de $O(m.d^2)$, où m est le nombre d'instances d'entraînement et d est le nombre d'attributs.

Intéprétabilité des arbres de décision. Les arbres de décision sont depuis longtemps reconnus comme des modèles de choix dans diverses applications en raison de leur interprétabilité. Ils permettent d'extraire facilement les règles qui ont conduit aux décisions prises par le modèle. Cependant, ils ne sont pas

toujours interprétables [IIMS20], c'est pourquoi des outils d'explicabilité des arbres de décision sont souvent nécessaires. Au chapitre 6 de ce mémoire, nous allons montrer que l'explicabilité des arbres de décision va au-delà de cette capacité d'extraction de règles de classement, et qu'ils offrent en réalité un pouvoir explicatif bien plus important. En résumé, les arbres de décision possèdent un potentiel explicatif considérable, (que nous détaillerons au chapitre 6) quand leur taille reste raisonnable.

3.2.2 Listes de décisions (*decision lists*)

La liste de décision est un autre modèle interprétable qui fut introduit pour la première fois par [Riv87]. Une liste de décision est un ensemble de règles de classification individuelles qui forment collectivement un classifieur. Contrairement à un ensemble de règles non ordonné, les listes de décision ont un ordre inhérent, ce qui facilite la classification. Lorsqu'une nouvelle instance doit être classée, les règles sont évaluées dans l'ordre, et la classe prédite est déterminée par la première règle qui couvre l'instance. Si aucune règle ne correspond, une règle par défaut est utilisée, qui prédit généralement la classe majoritaire dans le jeu de données d'entraînement.

Définition 40 (listes de décision). *Les listes de décision [Riv87] sont des multi-ensembles ordonnés de règles de la forme* $L = \langle t_1, c_1\rangle, \ldots, \langle t_m, c_m\rangle$*, où chaque* t_i $(i \in [m])$ *est un terme sur* $X_d = \{x_1, \cdots, x_d\}$*, et chaque* c_i *est une valeur booléenne dans* $\mathbb{B} = \{0, 1\}$*. Une instance d'entrée* $\boldsymbol{x} \in \mathbb{B}^d$ *est un modèle de* L *si la classe* c_i *de la première règle* t_i *qui correspond à* $\boldsymbol{x}$ *est positive. Par convention, dans la dernière règle,* t_m *est le terme vide* $\top$*. Formellement,* $L(\boldsymbol{x}) = c_j$*, où* $j = \operatorname{argmin}_{1\leq i\leq m} t_i(\boldsymbol{x}) = 1$*. La taille d'une liste de décision* L *est la somme des tailles des termes présents dans* L*.*

Exemple 25. *Un exemple d'une liste de décision sur* $X_4 = \{x_1, x_2, x_3, x_4\}$ *est donné par :*

$$L = \langle x_1 \wedge x_2, 1\rangle, \langle \overline{x_1}, 0\rangle, \langle x_3 \wedge x_4, 1\rangle, \langle \top, 0\rangle$$

Pour les instances $\boldsymbol{x}_1 = (1, 1, 0, 0)$ *et* $\boldsymbol{x}_2 = (0, 1, 0, 0)$*, nous avons* $L(\boldsymbol{x}_1) = 1$ *et* $L(\boldsymbol{x}_2) = 0$

Comme les attributs des listes de décisions forment toujours une séquence linéaire de décisions, la branche à vrai d'une décision pointe toujours vers 1 ou 0, jamais vers un autre nœud de décision. Comparées aux arbres de décision, les listes de décisions offrent une structure plus simple, mais la complexité des décisions autorisées à un nœud est plus grande.

Interprétabilité des listes de décision. L'interprétabilité des listes de décision est souvent considérée comme élevée. Étant donné leur structure, il est généralement simple de comprendre comment une instance est classée en parcourant la liste de haut en bas et en appliquant la première règle dont la condition est satisfaite. De plus, les règles utilisées dans les listes de décision sont souvent formulées de manière logique, ce qui facilite leur interprétation. Cependant, il convient de noter que les listes de décision deviennent plus complexes à mesure que le nombre de règles augmente. La présence de nombreuses règles peut rendre la liste difficile à suivre et à interpréter. Dans de tels cas, des techniques de simplification, telles que la sélection de règles les plus importantes ou la fusion de règles similaires, peuvent être utilisées pour améliorer l'interprétabilité.

3.2.3 Règles de décisions (*decision rules*)

Une règle de décision est une instruction simple de la forme **SI-ALORS**, comprenant une condition (ou antécédent) et une prédiction. Par exemple : SI une personne a plus de 18 ans ET SI elle a un permis de conduire valide (condition), ALORS elle peut louer une voiture (prédiction). On peut utiliser une seule règle de décision ou combiner plusieurs règles pour effectuer des prédictions. Les règles de

décision suivent une structure générale : SI certaines conditions sont remplies, ALORS une prédiction est faite. Les règles de décision sont considérées comme l'un des modèles de prédiction les plus interprétables. Leur structure SI-ALORS est semblable au langage naturel et à notre mode de pensée, à condition que les conditions soient construites à partir de caractéristiques compréhensibles, que la longueur de la condition soit courte et qu'il n'y ait pas trop de règles.

Une analogie pour comprendre un ensemble de règles de décision est de le considérer comme une forme de démocratie des règles, où chaque règle a un certain pouvoir de vote. Dans un tel ensemble, les règles peuvent soit s'exclure mutuellement, soit il existe des mécanismes de résolution des conflits, comme un vote à la majorité, qui peut être pondéré en fonction de la précision des règles individuelles ou d'autres mesures de qualité. Cependant, lorsque plusieurs règles s'appliquent, cela peut entraîner une perte d'interprétabilité. Il existe de nombreuses manières d'apprendre des règles de décision à partir de données. Dans cette section, nous présentons une méthode citée dans [Mol20]. L'algorithme est choisi pour couvrir un large éventail d'idées générales pour l'apprentissage des règles de décision, de sorte qu'ils représentent tous les trois des approches différentes.

OneR couvre toujours toutes les instances du jeu de données d'entraînement. Un modèle OneR est un arbre de décision avec un seul point de division. La partition n'est pas nécessairement binaire comme dans l'algorithme CART, mais dépend du nombre de valeurs de l'attribut. Lorsque deux attributs entraînent la même erreur totale, OneR résout les égalités en prenant soit le premier attribut avec l'erreur la plus faible, soit celui avec la p-value [LS20] la plus faible d'un test du χ^2.

Les règles de décision sont-elles réellement interprétables ? Lorsque l'on utilise un vote majoritaire pour déterminer l'étiquette d'une instance donnée et que la taille ou le nombre des règles augmente, la prédiction peut devenir difficile à comprendre pour un humain. Cela nous conduit à conclure que les règles de décision ne sont pas toujours interprétables.

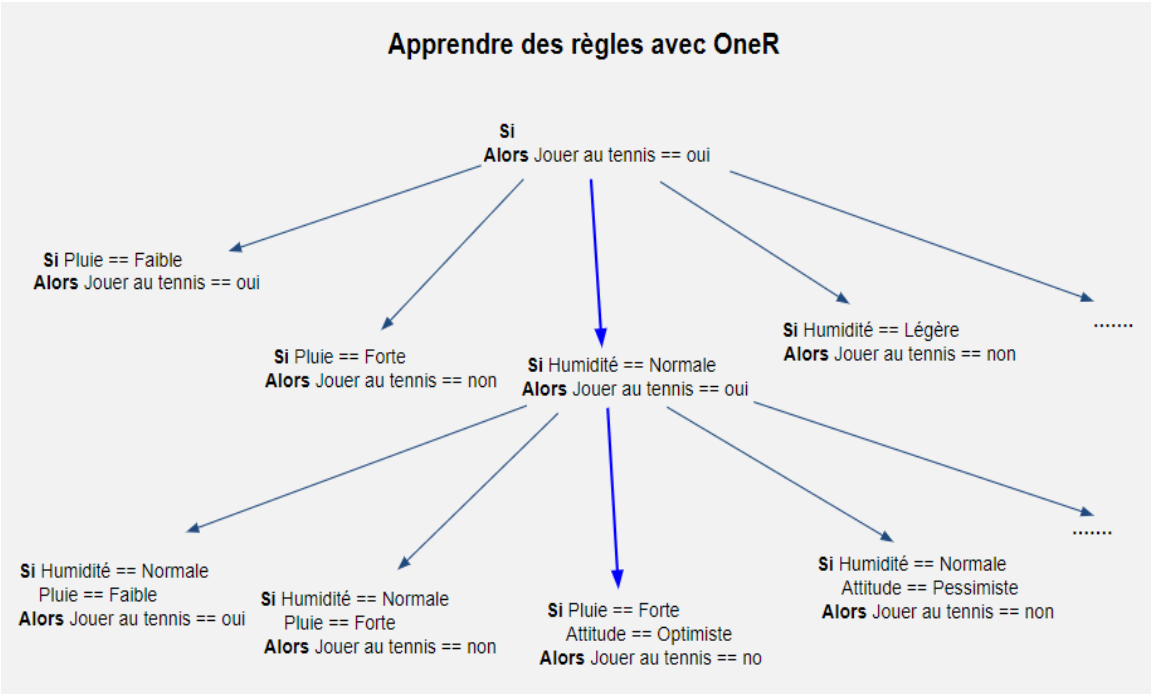


FIGURE 9 – Règles de décision.

Exemple 26. *La figure 9 présente un exemple des règles de décision apprises avec l'algorithme **OneR**. Le principe du schéma illustré à la figure 9 consiste simplement à construire les règles de décision. Débutons par choisir une condition que nous souhaitons utiliser comme variable principale pour prendre la décision. Si ensuite une condition aboutit à un résultat « non », alors on peut conclure que l'individu ne joue pas au tennis, et l'exploration d'autres conditions possibles s'arrête. Quand la condition conduit à un résultat « oui », nous poursuivons en explorant d'autres conditions jusqu'à obtenir un ensemble de règles qui favorisent la décision de jouer au tennis. Par exemple, une règle apprise indique que si la pluie est trop forte, alors l'individu ne jouera pas au tennis.*

3.3 Modèles ensemblistes

Les méthodes d'ensemble [Die00] sont des algorithmes d'apprentissage supervisé qui visent à former plusieurs classifieurs et à combiner leurs prédictions [Zho12] pour classer de nouveaux points de données. Une architecture courante des méthodes d'ensemble est illustrée à la figure 10. La méthode d'ensemble originale est la moyenne bayésienne. En général, une méthode d'ensemble utilise un seul algorithme d'apprentissage pour générer les modèles de base. Les méthodes d'ensemble sont souvent plus performantes que n'importe quel modèle de base, comme le confirme le modèle de forêts aléatoires par rapport au modèle d'arbre de décision, ainsi que d'autres modèles tels que XGBoost, Adaboost, etc. Nous rappelons maintenant brièvement deux types majeurs de techniques fréquemment utilisées pour construire des modèles d'ensemble : *le bagging et le boosting*.

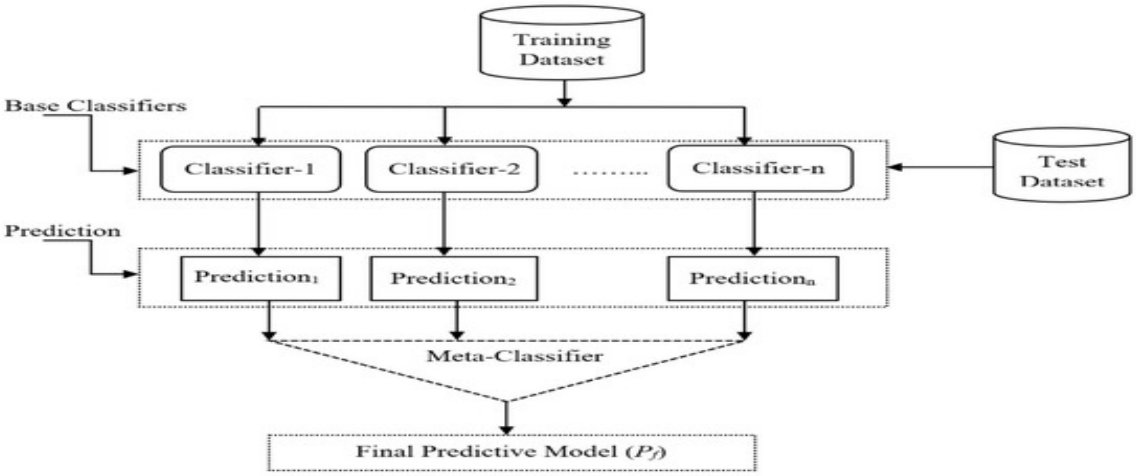


FIGURE 10 – Une architecture des méthodes d'ensemble (source : ensemble)

Bagging. Le bagging [Bre96], est une technique qui combine ébauchage et agrégation pour former un modèle d'ensemble. Cette méthode consiste à créer plusieurs sous-échantillons *bootstrap* à partir d'un échantillon de données donné. Un classifieur faible est construit pour chaque sous-échantillon *bootstrap*. Ensuite, les classifieurs des modèles sont agrégées à l'aide d'un algorithme spécifique afin de former un classifieur plus puissant. L'objectif est d'améliorer la stabilité et la précision des prédictions en exploitant la diversité des sous-échantillons *bootstrap*. La figure 11 illustre le principe du bagging appliqué sur une base de données et un modèle faible d'arbre de décision.

Boosting. Les méthodes de boosting constituent une famille d'algorithmes qui génèrent des classifieurs de base de manière séquentielle. Le principe général du *boosting* est le suivant : d'abord, un modèle faible

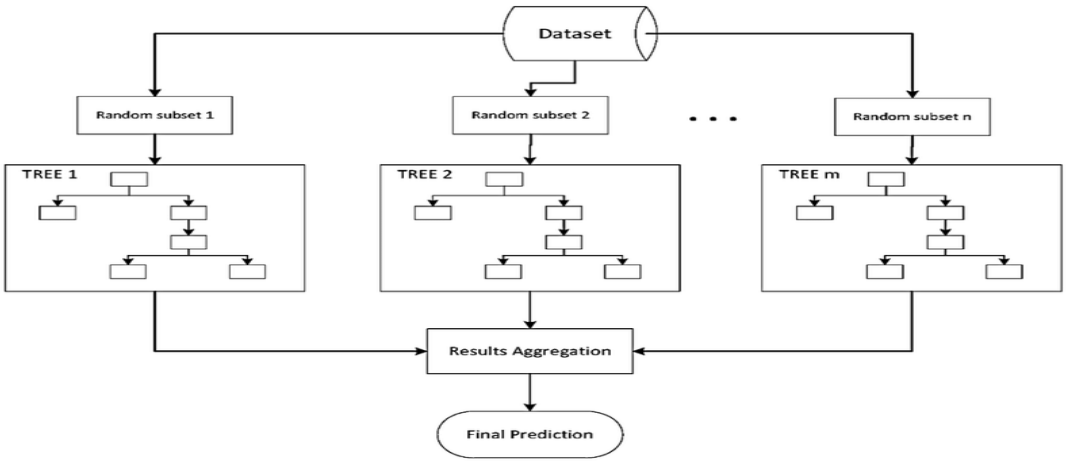


FIGURE 11 – Une architecture des méthodes d'ensemble (source : bagging)

est entraîné sur l'ensemble de données initial. Ensuite, la distribution des données est ajustée en fonction des prédictions du modéle de base. Les exemples mal prédits se voient attribuer des poids plus élevés afin de les mettre davantage en évidence lors des itérations suivantes. Ensuite, un nouveau modèle de base est entraîné en utilisant l'ensemble de données ajusté. Ce processus est répété jusqu'à atteindre le nombre d'itérations prédéfini. Enfin, les modèles de base sont combinés pour former un modèle final.

Il existe de nombreux algorithmes de *boosting et de bagging*, parmi lesquels AdaBoost (Adaptive Boosting) [FHT00], GBT (Gradient-Boosted Trees) [Fri00], Random Forest [Bre01], XGBoost (eXtreme Gradient Boosting) [CG16].

3.3.1 Forêts aléatoires

Les forêts aléatoires sont des algorithmes qui reposent sur la méthode de *bagging* et combinent plusieurs arbres de décision pour effectuer des prédictions. Une forêt aléatoire [Bre01] est essentiellement un ensemble d'arbres de décision individuels, où chaque arbre de décision est formé sur un sous-ensemble aléatoire des données d'entraînement. L'idée principale derrière les forêts aléatoires est de créer un modèle robuste et performant en combinant les prédictions de plusieurs arbres de décision. Chaque arbre de décision dans la forêt est construit de manière indépendante en utilisant un échantillon aléatoire des données d'entraînement et une sélection aléatoire des attributs à chaque nœud de l'arbre. Lorsqu'une prédiction est effectuée, chaque arbre de décision vote pour une classe ou une valeur de sortie, et la classe donnant la valeur avec le plus de votes est choisie comme prédiction finale de la forêt. Les forêts aléatoires sont connues pour leur capacité à gérer des ensembles de données de grande taille avec des attributs complexes. Elles sont robustes au bruit et aux valeurs aberrantes. De plus, les forêts aléatoires peuvent également fournir des estimations de l'importance des attributs, ce qui permet de comprendre l'influence de chaque variable sur les prédictions du modèle.

Définition 41 (forêt aléatoire). *Une* forêt aléatoire *sur X_d est un ensemble $F = \{T_1, \cdots, T_m\}$, où chaque T_i $(i \in [m])$ est un arbre de décision sur X_d, et tel que la valeur $F(\boldsymbol{x})$ est donnée par*

$$F(\boldsymbol{x}) = \begin{cases} 1 & \textit{si } \frac{1}{m}\sum_{i=1}^{m} T_i(\boldsymbol{x}) > \frac{1}{2} \\ 0 & \textit{sinon.} \end{cases}$$

La taille *de F est donnée par $|F| = \sum_{i=1}^{m} |T_i|$, où $|T_i|$ est le nombre de nœuds présents dans T_i. La classe des arbres de décision sur X_d est désignée par $\mathtt{DT}_d$, et la classe des forêts aléatoires avec au plus*

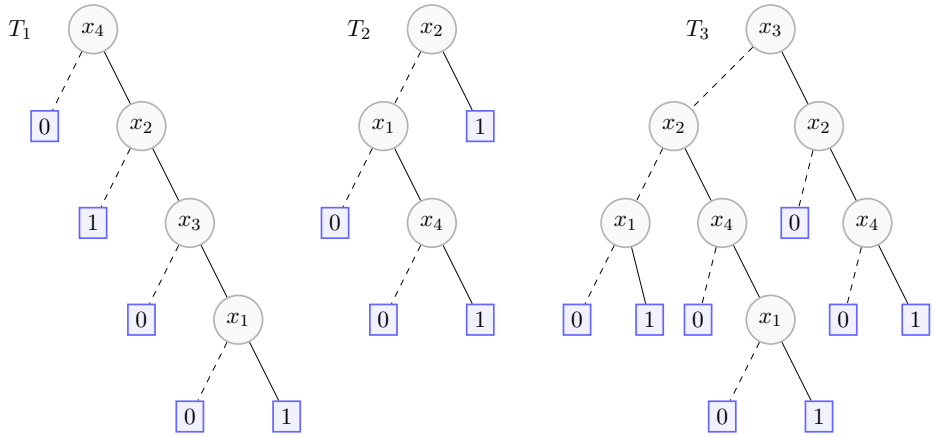


FIGURE 12 – Une forêt aléatoire F pour décider d'accorder ou non un prêt à un client.

m arbres de décision (avec $m \geq 1$) sur $\texttt{DT}_d$ est désignée par $\texttt{RF}_{d,m}$. Enfin, $\texttt{RF}_d$ est l'union de tous les ensembles $\texttt{RF}_{d,m}$ pour $m \geq 1$. $\texttt{RF}_d = \bigcup_{m \geq 1} \texttt{RF}_{d,m}$ *et* $\texttt{RF} = \bigcup_{d \geq 1} \texttt{RF}_d$.

Un exemple illustrant le concept des forêts aléatoires pour la classification est présenté à la figure 12. Cette forêt a été apprise à partir d'un ensemble de données contenant des informations sur des clients :

— x_1 (situation professionnelle) : « le client n'est pas en contrat CDI »
— x_2 (âge) : « le client a plus de 50 ans »
— x_3 (niveau de revenu) : « les revenus annuels du client sont inférieurs à 35K € »
— x_4 (fiabilité du client) : « le client n'a pas remboursé un prêt précédent »

Maintenant, prenons des informations sur un client traduites par l'instance $\boldsymbol{x} = (1, 0, 0, 1)$. La forêt aléatoire de la figure 12 classe cette instance comme positive, car la majorité des arbres ont classé l'instance comme positive ($T_1(\boldsymbol{x}) = 1, T_2(\boldsymbol{x}) = 0, T_3(\boldsymbol{x}) = 1$). Cela indique que le prêt a été accepté.

Interprétabilité des forêts aléatoires. Les forêts aléatoires sont généralement considérées comme des modèles « boîte noire » en raison de leur complexité et de la nature de l'ensemble de modèles.

3.3.2 Boosting d'arbres

Le boosting d'arbres est une méthode très populaire qui permet d'améliorer les performances des arbres de décision en les combinant de manière séquentielle. Cela permet de construire des modèles plus puissants, mais aussi plus complexes. Parmi les systèmes de boosting d'arbres couramment utilisés, nous trouvons GBT (*Gradient Boosting Tree*), XGBoost [CG16] et LightGBM [KMF+17]. Ces méthodes sont largement adoptées. Elles présentent des similitudes avec les forêts aléatoires, qui combinent également plusieurs arbres de décision. Cependant, les méthodes de *boosting* d'arbres peuvent être plus rapides et offrir des avantages en termes de performances et d'efficacité, le boosting d'arbres est une technique d'apprentissage automatique itérative où des arbres de décision simples sont construits de manière séquentielle, en se concentrant sur les échantillons mal classés par les arbres précédents. Chaque nouvel arbre est ajouté pour corriger les erreurs commises par les arbres précédents et pour améliorer la performance globale du modèle. Finalement, les prédictions des différents arbres sont combinées pour produire la prédiction finale.

Définition 42 (**arbres boostés** [FS95, SF12])**.** *Les* arbres boostés *sont une combinaison convexe d'arbres de décision de la forme $B = \langle T_1, \alpha_1 \rangle, \ldots, \langle T_m, \alpha_m \rangle$, où chaque $T_i, (i \in [m])$ est un arbre de décision*

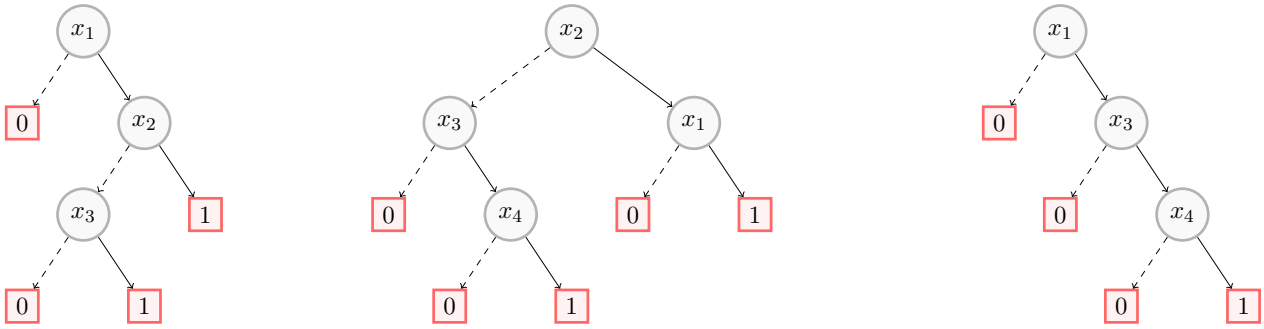


FIGURE 13 – Un modèle d'arbres boostés B pour décider d'accorder ou non un prêt à un client. Les poids des arbres sont respectivement de $0,5$, $0,25$ et $0,25$.

et $\boldsymbol{\alpha} = \{\alpha_1, \cdots, \alpha_m\}$ est une combinaison convexe de coefficients.[5] *Comme avec les forêts aléatoires, les décisions prises par les arbres boostés sont déterminées par un vote majoritaire pondéré :*

$$B(\boldsymbol{x}) = \begin{cases} 1 & \textit{si } \sum_{i=1}^{m} \alpha_i T_i(\boldsymbol{x}) > \frac{1}{2} \\ 0 & \textit{sinon.} \end{cases}$$

La taille de B notée $|B|$ est à nouveau donnée par la somme des tailles de ses arbres.

Un exemple illustrant le concept des arbres boostés pour la classification binaire est présenté à la figure 13. Les informations sur des clients :

— x_1 (situation professionnelle) : « le client n'est pas en contrat CDI »
— x_2 (âge) : « le client a plus de 50 ans »
— x_3 (niveau de revenu) : « les revenus annuels du client sont inférieurs à 35K € »
— x_4 (fiabilité du client) : « le client n'a pas remboursé un prêt précédent »

Le modèle d'arbres boostés de la figure 13 classe instance l'instance $\boldsymbol{x} = (1, 1, 1, 1)$ comme positive, car tous les arbres votent 1 et donc $\sum_{i=1}^{m} \alpha_i T_i(\boldsymbol{x}) = 1$.

Interprétabilité des arbres boostés. Les modèles de *boosting* basés sur des arbres de décision sont généralement considérés comme moins interprétables que les arbres de décision individuels. Cela est dû à plusieurs raisons. Premièrement, le processus de *boosting* [CG16] combine de nombreux arbres de décision, ce qui peut rendre difficile l'interprétation des prédictions finales. Deuxièmement, le *boosting* utilise des techniques telles que l'ajustement des poids des exemples et la création de sous-ensembles de données, ce qui ajoute de la complexité. Enfin, la combinaison de nombreux arbres peut rendre difficile l'identification des attributs spécifiques qui influencent les prédictions. Certaines variantes du *boosting*, comme XGBoost et LightGBM, offrent des fonctionnalités pour visualiser l'importance des caractéristiques, ce qui peut aider à une certaine interprétabilité. Mais de manière générale, les modèles de *boosting* sont souvent considérés comme des « boîtes noires » en termes d'interprétation.

3.3.3 Quelques statistiques sur les arbres de décision et les ensembles d'arbres

Dans ce travail, les arbres de décision et les modèles d'ensemble occupent une place prépondérante. les arbres de décisions, bien que réputés pour leur interprétabilité, ne sont pas à l'abri de certaines limitations en pratique. Il est donc essentiel de se pencher sur les statistiques liées à ces modèles pour mieux

5. En d'autres termes, $\alpha_i \geq 0$ pour tout $i \in [m]$ et $\sum_{i=1}^{m} \alpha_i = 1$.

Benchmark			Decision trees			Random forest		
name	N	I	*acc*	$\|T\|$	RD	*acc*	$\|F\|$	DEPTH
compas	11	6172	65.12	550	10.94 ($\pm$2.18)	67.06	19032	17.63
gisette	5000	7000	93.43	151	25.15 ($\pm$10.95)	96.62	17959	26.87
farm-ads	54876	4143	87.13	222	21.59 ($\pm$17.55)	90.19	20516	53.05
mnist49	784	13782	95.28	231	18.91 ($\pm$5.02)	98.5	15444	20.2
mnist38	784	13966	95.78	209	17.47 ($\pm$4.67)	98.74	14203	18.24
bank	20	41188	88.77	2116	14.89 ($\pm$2.39)	91.05	83828	27.27
christine	1636	5418	61.32	322	15.57 ($\pm$8.42)	71.89	31105	26.32

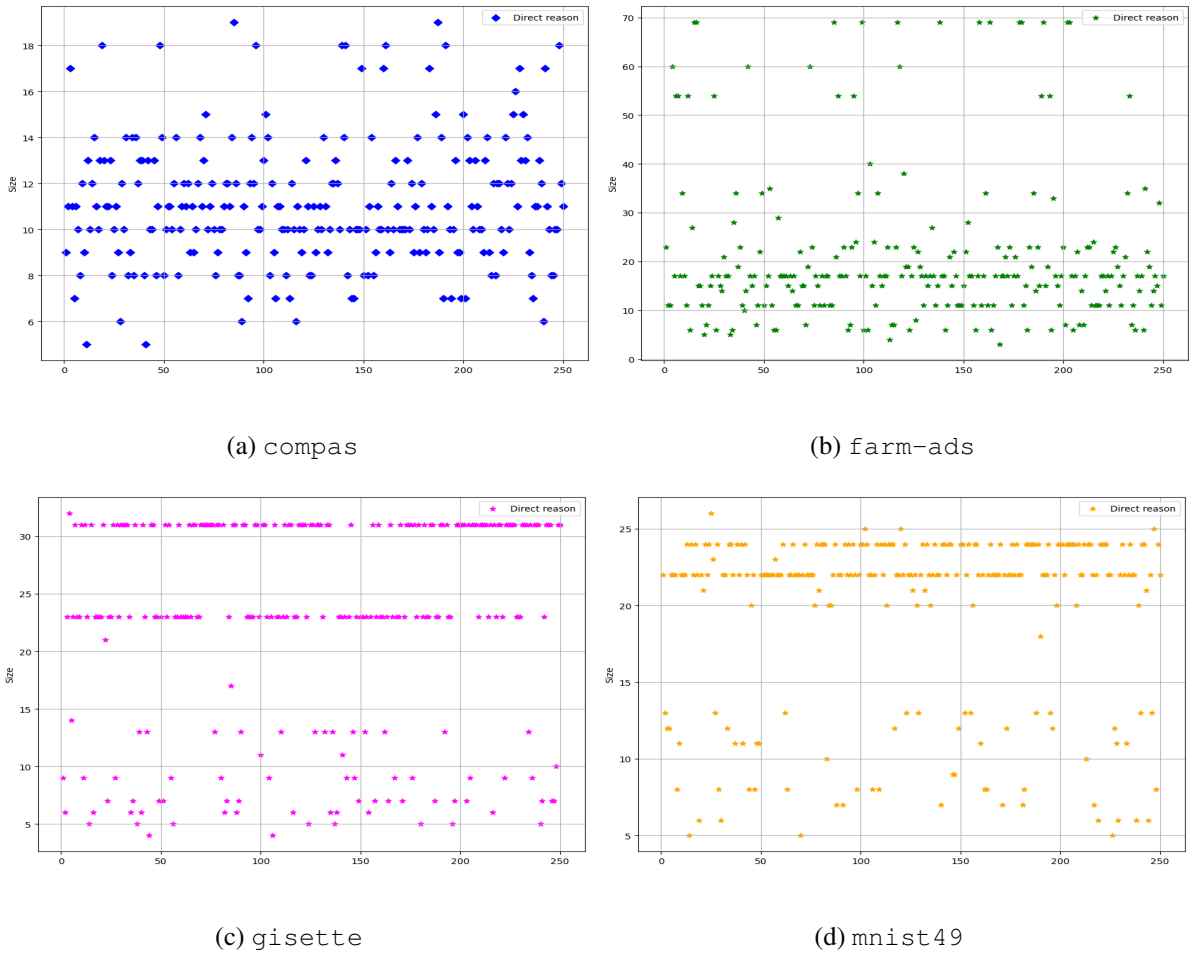
TABLE 3 – Quelques statistiques sur 7 benchmarks pour les arbres de décision et les forêts aléatoires.

comprendre le besoin vital d'explication. Dans cette section, nous examinerons notamment les statistiques concernant la profondeur et la taille des arbres générés par [PVG+11]. Il est courant de constater que ces arbres peuvent devenir excessivement profonds ou comporter un grand nombre de nœuds, ce qui peut compromettre leur interprétabilité. Un exemple concret est fourni par un arbre de décision entraîné sur le benchmark *Gisette*, montrant qu'il est difficile de comprendre l'explication du chemin [IIMS20] (appelée aussi raison directe [ABB+22d]) en raison de sa taille très élevée. Les forêts aléatoires, quant à elles, génèrent souvent un grand nombre d'arbres dans leur ensemble pour garantir une bonne précision. Certains de ces arbres peuvent être trop grands et trop profonds, et cet ensemble de plusieurs arbres peut être plus difficile à interpréter qu'un modèle unique.

En somme, cette exploration des statistiques liées aux arbres de décision et aux forêts aléatoires nous permettra de mieux saisir les limites de l'interprétabilité de ces modèles et de mettre en évidence la nécessité de développer des mécanismes explicatifs pour les rendre plus compréhensibles et utiles dans des applications réelles.

Quelques statistiques. Le tableau 3 présente certaines statistiques sur les modèles d'arbres de décision et les forêts aléatoires qui ont été appris sur un ensemble de 7 *benchmarks* à l'aide de leur implémentation dans [PVG+11]. La colonne la plus à gauche donne le nom du benchmark, N et I donnent respectivement le nombre de variables et le nombre d'instances de chaque benchmark dans le tableau. Les colonnes $|T|$ et $|F|$ donnent respectivement la précision des arbres de décision (resp. forêts aléatoires), les tailles des arbres de décision et celles des forêts. La colonne RD indique la taille moyenne des raisons directes (explication du chemin [IIMS20]), tandis que DEPTH indique la profondeur maximale moyenne des arbres de décision présents dans les forêts aléatoires.

En analysant les tailles des arbres de décision, on constate qu'ils sont généralement de grande taille, ce qui implique davantage de nœuds et de règles apprises. Cette augmentation de la taille des arbres peut également rendre leur interprétation plus complexe. De plus, la taille moyenne des raisons directes (RD) pour chaque *benchmark* est considérable, dépassant en moyenne $10,5$. Cette observation souligne la difficulté potentielle à expliquer des arbres aussi grands et profonds. Il est à noter que cette taille moyenne dépasse largement la limite cognitive de 7 [Mil56]. Ces observations mettent en évidence la nécessité de développer des techniques d'explication, qu'elles soient abductives ou probabilistes, pour rendre ces arbres plus compréhensibles malgré leur complexité. En ce qui concerne les forêts aléatoires, nous avons principalement rapporté des résultats concernant la taille des forêts et la profondeur maximale moyenne des arbres qui les composent. Bien que les forêts aléatoires se révèlent très performantes en termes de précision, leur taille est souvent très importante, notamment dans le cas de *farm-ads*. De plus, la profondeur maximale moyenne des arbres de la forêt est grande. Ces arbres profonds ne sont pas



(a) `compas` (b) `farm-ads`

(c) `gisette` (d) `mnist49`

FIGURE 14 – Scatter plot des tailles des raisons directes pour 4 benchmarks.

interprétables, et donc on a besoin de pouvoir calculer des explications (abductives ou probabilistes) pour les rendre plus compréhensibles.

La figure 14 présente un nuage de points représentant les explications du chemin [IIMS20] (ou *raisons directes* [ABB+22d]) calculées pour 250 instances issus de 4 benchmarks différents. On peut observer que, pour tous les benchmarks, la taille des raisons directes dépasse 7, ce qui rend ces explications difficilement interprétables.

La figure 14 présente un un nuage de points. Chaque point dans ce nuage correspond à une explication du chemin [IIMS20], également appelée *raisons directes* [ABB+22d], calculée pour un total de 250 instances. Ces instances proviennent de quatre benchmarks distincts. L'observation clé que nous pouvons tirer de ce graphique est que, pour les 4 benchmarks considérés, la taille des raisons directes associées à chaque instance est supérieure à 7. Cette taille représente le nombre de caractéristiques inclus dans chaque explication. Le fait que ces explications soient constamment de grande taille, dépassant 7, suggère qu'elles sont complexes et difficiles à interpréter pour un les humains. Par conséquent, cette constatation met en lumière la nécessité du calcul d'autres types d'explications, notamment abductives et probabilistes. Dans l'espoir de fournir des explications plus concises et compréhensibles.

En résumé, la figure 14 montre que les raisons directes sont souvent trop longues pour être facilement comprises, ce qui conduit à la recherche de méthodes d'explication plus efficaces et accessibles, telles que les explications abductives et probabilistes.

3.4 Conclusion

Nous avons décrit différents modèles d'apprentissage automatique et des techniques utilisées pour améliorer leurs performances. Les modèles symboliques, tels que les arbres de décision, les listes de décision et les règles de décision, ont été présentés, ainsi que les modèles basés sur des arbres et les techniques de *boosting* et de *bagging*. Les modèles tels que les arbres boostés et les forêts aléatoires sont reconnus pour leurs performances prédictives. Cependant, ils sont souvent considérés comme des « boîtes noires » en raison de leur faible niveau d'interprétabilité, c'est-à-dire qu'il est difficile de comprendre et d'expliquer comment ces modèles prennent leurs décisions. En revanche, les modèles symboliques tels que les arbres de décision sont plus interprétables, c'est-à-dire qu'il est plus facile de comprendre les règles et les décisions prises par ces modèles. Ces modèles symboliques ont généralement des performances de prédiction moins bonnes que les modèles basés sur les arbres boostés et les forêts aléatoires. Malgré cette limitation, les modèles symboliques restent un choix privilégié dans les applications sensibles où l'interprétabilité est primordiale, car ils permettent aux utilisateurs de comprendre le processus décisionnel et d'expliquer les résultats obtenus.

4

Explicabilité des prédictions en apprentissage automatique

Sommaire

4.1 Panorama des explications 60
4.1.1 Introduction 60
4.1.2 Formes d'explications 61
4.1.3 Variabilité des formes d'explications 62
4.1.4 Propriétés des explications 62
4.2 Méthodes agnostiques locales « orientées boîtes noires » 64
4.2.1 LIME 64
4.2.2 SHAP (*SHapley Additive exPlanations*) 66
4.2.3 Anchors 69
4.2.4 Limites et faiblesses des approches agnostiques de l'état de l'art 71
4.3 Méthodes formelles orientées modèles 71
4.3.1 Explications abductives 72
4.3.2 Explications contrastives 73
4.3.3 Notes complémentaires 73
4.4 Conclusion 74

Avec l'essor croissant de l'apprentissage automatique (dont l'apprentissage profond), les applications de cette technologie se sont diversifiées et généralisées dans de nombreux domaines tels que la santé, la finance, la cybersécurité, le droit, les transports. Cependant, la complexité de ces modèles rend difficile l'interprétation de leurs décisions. Cette opacité soulève des inquiétudes quant à la transparence et à la compréhensibilité de ces modèles, et des conséquences néfastes, parfois inattendues, de leur utilisation ont été documentées. Le scandale du logiciel **COMPAS** [6] en 2016 a mis en évidence le biais racial d'un logiciel utilisé pour prédire le risque de récidive des détenus, mettant en lumière les problèmes juridiques et éthiques liés à l'opacité des algorithmes utilisés.

Dans les domaines sensibles tels que la médecine ou le droit, les acteurs choisissent souvent des modèles symboliques moins performants mais plus interprétables, comme brièvement expliqué au chapitre 3. Cependant, pour d'autres domaines, sacrifier les performances prédictives n'est pas considéré comme

6. https ://www.propublica.org/article/how-we-analyzed-the-compas-recidivism-algorithm

une option acceptable. Par conséquent, il existe un intérêt certain pour comprendre l'origine des prédictions sans compromettre la précision des prédictions, ce qui a conduit au développement du domaine de l'IA explicable (XAI).

4.1 Panorama des explications

Dans cette section, nous explorons les différentes dimensions du domaine de l'**XAI** à travers les explications fournies pour les systèmes d'IA. Ces explications prennent différentes formes, allant des explications textuelles aux explications graphiques. Il n'y a pas de consensus clair sur la forme que les explications doivent prendre, ce qui entraîne une variabilité des approches. Néanmoins, la capacité de *raisonner* à partir des explications est essentielle pour les utilisateurs afin de comprendre et de collaborer efficacement avec les systèmes d'IA. En intégrant les explications dans le processus de raisonnement, les utilisateurs peuvent tirer parti de ces informations pour prendre des décisions éclairées et améliorer la confiance dans les résultats fournis par les systèmes d'IA.

4.1.1 Introduction

Le domaine de l'IA explicable (XAI) a émergé en tant que champ de recherche distinct [Mol20, GMR+19, Mil19, LRMM15], principalement en raison de la montée en puissance de l'apprentissage automatique et de l'inclusion du droit à l'explication dans le règlement général sur la protection des données de l'Union Européenne. Au cours des dernières années, de nombreuses méthodes et approches ont été proposées pour expliquer les modèles d'apprentissage automatique [AB18, DK17, RSG16, LL17, RSG18, Rud18]. Ces méthodes diffèrent les unes des autres. Citons les méthodes **Grad-CAM** qui génèrent des explications visuelles sous forme de cartes de chaleur des zones d'intérêt dans une image pour expliquer les prédictions des modèles d'apprentissage profond [SDV+16, CSHB17]. D'autres méthodes d'interprétation des modèles fournissent des explications locales [RSG16, LL17] ou globales en identifiant les attributs importants pour les prédictions, tandis que certaines méthodes génèrent des règles compréhensibles pour expliquer les décisions du modèle [RSG18]. Des méthodes basées sur la logique propositionnelle ont également été proposées pour calculer des explications formelles avec des garanties formelles solides [MS23].

Les recherches dans le domaine ont mis en évidence l'importance de prendre en compte plusieurs aspects lors de l'évaluation de la qualité des explications. Selon [NTP+22], les critères de qualité incluent la fidélité de l'explication à représenter le raisonnement interne du modèle, la concision de l'explication et sa compréhensibilité pour les humains, ainsi que la cohérence de l'explication avec les connaissances du domaine d'application. La fidélité de l'explication est essentielle pour assurer une représentation précise du modèle, tandis que la concision est importante pour faciliter la compréhension par les utilisateurs finaux, comme souligné par Miller [Mil56]. De plus, il est crucial que l'explication soit cohérente avec les connaissances préexistantes dans le domaine. Cependant, malgré ces critères, il n'existe pas de définition générale de ce qui constitue la meilleure explication possible, comme noté dans [LEC+20]. Cette absence de définition claire souligne la complexité du domaine de recherche et l'importance de poursuivre les travaux dans ce domaine pour développer des normes et des critères d'évaluation robustes pour les explications. La forme de l'explication jouant un rôle crucial dans sa compréhension, nous allons lister les principales formes d'explications.

4.1.2 Formes d'explications

La forme de l'explication fournie à l'utilisateur joue un rôle crucial dans la communication des informations, la compréhension et la persuasion. Il est important de choisir une forme d'explication appropriée en fonction du public cible, du contexte d'utilisation et des objectifs de communication. Les explications peuvent revêtir différentes formes, telles que des explications textuelles, des ensembles de mots clés, ou des présentations graphiques utilisant des visualisations telles que des cartes conceptuelles (voir [DQA+20]).

Explications visuelles. Les explications visuelles sont particulièrement pertinentes dans le contexte de la classification d'images, où elles permettent de comprendre visuellement les raisons qui ont conduit à une certaine prédiction. Par exemple, dans la classification d'images médicales, les explications visuelles peuvent mettre en évidence les zones d'intérêt ou les structures anatomiques qui ont été utilisées par le modèle pour identifier une pathologie spécifique. Dans le domaine de la détection d'objets, les explications visuelles peuvent aider à comprendre les régions de l'image qui ont conduit à la détection d'un objet particulier (voir [CSHB17]). Les explications visuelles sont également utilisées dans d'autres domaines de la classification, tels que la reconnaissance de texte, la classification de documents, la classification de vidéos, etc. Dans tous ces cas, les explications visuelles permettent d'apporter une compréhension plus fine et intuitive des décisions prises par les modèles de classification, et renforcer ainsi la confiance et l'interprétabilité des résultats obtenus.

Explications textuelles. Les explications textuelles sont très utiles dans les modèles de classification de textes. Elles peuvent prendre la forme de mots clés, de phrases ou de paragraphes pour expliquer les raisons derrière les décisions prises ou non prises par le modèle. Par exemple, dans la classification de texte, l'explication peut être un ensemble de mots clés ou une phrase expliquant pourquoi le texte appartient à une catégorie plutôt qu'à une autre. Les explications textuelles sont extrêmement utiles dans de nombreux cas pratiques.

Explications à base logique. Une approche basée sur des concepts et des règles logiques peut être utilisée pour fournir des explications formelles et rigoureuses des décisions prises par les modèles d'apprentissage automatique [MS23]. En utilisant des concepts de logique formelle, comme la logique propositionnelle ou la logique du premier ordre, cette approche requiert de construire une explication du modèle. Ainsi, des explications claires, précises et compréhensibles sont obtenues à partir de cette représentation, ce qui est utile pour valider, interpréter et vérifier les décisions prises par les modèles d'apprentissage automatique. Ces explications sont particulièrement bénéfiques dans des domaines à haut risque tels que la sécurité, la médecine et la finance, où la transparence et la intéprétabilité des décisions sont essentielles.

La nature des explications dépend aussi de la question auxquelles elles répondent : pourquoi tel classement (explications abductives) ou pourquoi pas tel autre classement (explications contrastives).

Abduction. L'abduction vise à trouver la meilleure explication possible pour une instance donnée c'est-à-dire déterminer pourquoi l'instance est classé comme elle est classée, en identifiant les causes les plus probables. Par exemple, dans le contexte de la classification de texte, l'abduction peut être utilisée pour expliquer pourquoi un texte a été classé dans une catégorie spécifique plutôt que dans une autre.

Contraste. Le contraste vise à mettre en évidence les différences entre deux décisions qui peuvent être prises pour la même instance. Il cherche à déterminer pourquoi ne pas prendre une autre décision que la décision prise, en soulignant les causes qui peuvent balancer vers l'autre décision. Par exemple, dans le domaine des véhicules autonomes, le contraste peut être utilisé pour trouver une paire action-valeur qui conduit à un changement de la décision prise (prédiction), en identifiant les facteurs qui ont contribué à cette alternative de décision.

Les explications des modèles d'apprentissage automatique abordent les questions « Pourquoi ? » et « Pourquoi pas ? ». Les travaux de [INAMS20b] démontrent qu'il existe une relation formelle rigoureuse entre les explications abductives et contrastives. Plus précisément, il existe une forme de dualité entre les explications contrastives et abductives.

4.1.3 Variabilité des formes d'explications

Le domaine de l'intelligence artificielle explicable (XAI) a connu une croissance rapide. Cependant, un problème récurrent et reconnu dans ce domaine est le manque de consensus en ce qui concerne la terminologie utilisée. Chaque nouvelle contribution semble introduire sa propre version (souvent intuitive) des termes tels que « explication » et « interprétation ». Cette absence de consensus entrave la consolidation des progrès réalisés dans le domaine pour répondre aux exigences scientifiques et réglementaires. Bien qu'il y ait un consensus général sur l'importance de rendre les systèmes d'intelligence artificielle explicables et/ou interprétables, il n'y a pas de consensus clair sur ce que signifient réellement ces termes. Les travaux de Preece [PHB+18] soutiennent que ce manque de consensus est dû à l'existence de plusieurs communautés d'acteurs distinctes. Bien que ces communautés partagent des préoccupations générales, elles diffèrent dans leurs intentions et leurs exigences en matière d'explicabilité et d'interprétabilité.

Nous notons également que la variabilité des formes des explications pose un problème d'absence de consensus car il n'y a pas d'accord clair sur la forme que les explications doivent prendre. Les chercheurs et les utilisateurs utilisent différentes approches pour fournir des explications aux systèmes d'intelligence artificielle. En l'absence de consensus, il est difficile de définir des normes et des critères de qualité universellement acceptés. Cela peut entraîner une fragmentation de la communauté de recherche et rendre difficile la comparaison et l'évaluation des différentes approches d'explicabilité. À titre d'exemple, un expert peut trouver qu'il est plus intéressant de classer les attributs selon leur influence sur la décision prise, tandis que d'autres pourrait en trouver qu'il plus intéressant de fournir un impliquant de la décision prise par le modèle.

4.1.4 Propriétés des explications

Présentons maintenant quelques propriétés qui peuvent servir de critères pour évaluer la qualité d'une méthode d'explication ou d'une explication donnée [RSB18, Mol20].

Propriétés des méthodes d'explication.

— **La transparence :** la transparence d'une méthode d'explication se réfère à la manière dont elle utilise l'analyse directe du modèle d'apprentissage automatique, y compris ses paramètres. Les méthodes d'explication basées sur des modèles interprétables tels que la régression linéaire sont considérées comme transparentes car elles offrent une compréhension directe du modèle. En revanche, les méthodes qui se basent uniquement sur les entrées et les prédictions ont une transparence limitée. La transparence peut varier en fonction du contexte, avec une transparence élevée

permettant d'utiliser plus d'informations pour les explications, tandis qu'une transparence réduite rend la méthode plus portable et adaptable à différents modèles.

— **Le pouvoir expressif :** le pouvoir expressif est lié à la structure des explications que la méthode est capable de générer. Une méthode d'explication peut générer des règles, des ensembles de règles, une somme pondérée, ou une autre forme.

— **La complexité :** la complexité calculatoire fait référence à la difficulté de calcul de la méthode utilisée pour générer une explication. Cette caractéristique est essentielle à prendre en compte lorsque le temps de calcul est un facteur limitant dans la génération des explications.

Propriétés des explications individuelles.

— **Cohérence :** une explication d'une instance doit être cohérente avec la prédiction des instances du modèle d'apprentissage automatique. Un utilisateur n'acceptera pas une explication qui explique simultanément une instance classée positivement et une instance classée négativement, car dans ce cas, l'explication manque de cohérence.

— **Fidélité et précision :** la fidélité est un critère crucial d'une explication, car une explication de faible fidélité est peu utile pour comprendre le modèle d'apprentissage automatique. La précision et la fidélité sont étroitement liées : si le modèle est hautement précis et que l'explication est fidèle, alors l'explication sera également précise. Certaines explications ne présentent qu'une fidélité locale, ce qui signifie qu'elles ne correspondent étroitement à la prédiction du modèle que pour un sous-ensemble de données ou même pour une seule instance.

— **Stabilité et robustesse :** alors que la cohérence compare les explications entre les instances classées différemment, la stabilité compare les explications entre des instances similaires classées de la même manière. Une stabilité élevée signifie que de légères variations dans l'une des instances n'entraînent pas de modifications substantielles dans l'explication. Un manque de stabilité peut être dû à la sensibilité de la méthode d'explication ou à la difficulté du modèle à comprendre les données d'apprentissage.

— **Compréhensibilité :** ici, la question posée est la suivante : dans quelle mesure les humains comprennent-ils les explications ? La compréhensibilité est difficile à définir et à mesurer, mais elle est extrêmement importante. La compréhensibilité dépend du public cible. Des idées pour mesurer la compréhensibilité incluent la taille de l'explication (le nombre d'attributs mobilisés), la clarté du langage utilisé et la cohérence avec les connaissances préexistantes.

Raisonner à partir des explications. Raisonner à partir des explications consiste à utiliser les informations fournies par les explications pour comprendre les facteurs qui ont contribué à une prédiction spécifique ou à un comportement général du modèle. En analysant les explications, on peut découvrir des relations causales, des raisons ou des tendances dans les données. Cela peut aider à identifier les erreurs du modèle, à détecter les biais potentiels et à améliorer la confiance et l'acceptation des résultats. Raisonner à partir des explications permet également de formuler des hypothèses sur les mécanismes sous-jacents du modèle et de générer de nouvelles idées ou stratégies pour résoudre des problèmes complexes. En fin de compte, la capacité à raisonner à partir des explications renforce notre capacité à interagir de manière plus éclairée et responsable avec les systèmes d'IA.

Dans la section suivante, nous présenterons un état de l'art des méthodes de génération d'explications. Ces méthodes se distinguent les unes des autres par leur nature, mais elles ont en commun, comme mentionné dans l'introduction générale de ce mémoire, leur capacité à générer des explications locales.

4.2 Méthodes agnostiques locales « orientées boîtes noires »

Dans cette section, nous abordons la problématique d'explications des prédictions de l'apprentissage automatique en utilisant des approches indépendantes du modèle, connues sous le nom d'approches agnostiques. En considérant les modèles d'apprentissage automatique comme des boîtes noires, ces approches offrent une flexibilité essentielle dans le choix des modèles, des explications et des représentations, améliorant ainsi le processus de débogage, la comparaison des modèles et les interfaces utilisateur pour une variété d'utilisateurs et de modèles. Nous présentons également les principaux défis posées de ces méthodes et passons en revue certaines approches d'explications agnostiques récemment introduites, telles que LIME [RSG16], SHAP [LL17] et Anchors [RSG18], qui tentent de relever ces défis.

Méthodes agnostiques locales. Les méthodes agnostiques locales sont des approches d'interprétation de modèles d'apprentissage automatique qui se concentrent sur la génération d'explications pour des prédictions spécifiques, sans nécessiter de connaissances sur la structure interne du modèle. Contrairement aux méthodes globales qui cherchent à expliquer le modèle dans son ensemble, les méthodes agnostiques locales se concentrent sur l'explication des décisions prises pour des instances individuelles, offrant ainsi une explication plus ciblée. Ces méthodes sont particulièrement utiles lorsque les modèles sont complexes et considérés comme des « boîtes noires », c'est-à-dire qu'il est difficile de comprendre leur fonctionnement interne. Les exemples de modèles complexes incluent les réseaux de neurones profonds et ainsi que d'autres modèles opaques qui peuvent fournir de très bonnes performances prédictives mais qui sont difficiles à interpréter.

Contrairement aux méthodes d'explication qui nécessitent une connaissance détaillée de la structure interne du modèle, les méthodes agnostiques locales se basent uniquement sur les données d'entrée et les sorties prédites pour générer des explications. Elles tentent de répondre à des questions telles que « Pourquoi ce modèle a-t-il prédit cela pour cette instance spécifique ? » en utilisant des techniques telles que la perturbation des attributs, l'analyse de la sensibilité ou l'exploration des voisinages. En utilisant ces méthodes agnostiques locales, il est possible de comprendre les raisons spécifiques derrière les prédictions du modèle pour des exemples individuels, ce qui permet aux utilisateurs de gagner un peu en transparence et en confiance dans les résultats du modèle. Les explications fournies peuvent aider à identifier les attributs qui ont le plus d'influence sur les prédictions, à détecter des biais ou des erreurs potentielles, et à obtenir une compréhension plus approfondie du fonctionnement du modèle.

En résumé, les méthodes agnostiques locales constituent une approche pratique très intéressante pour interpréter des modèles d'apprentissage automatique complexes et opaques, en fournissant des explications ciblées pour des prédictions spécifiques sans nécessiter une connaissance détaillée de la structure interne du modèle.

4.2.1 LIME

LIME (Local Interpretable Model-Agnostic Explanations) [RSG16]. LIME est un algorithme utilisé pour expliquer les prédictions de n'importe quel classifieur ou régresseur en les approximant localement avec un modèle interprétable. Il modifie un échantillon de données en ajustant les valeurs des attributs et observe l'impact qui en résulte sur la sortie. Il joue le rôle d'un « explicateur » pour expliquer les prédictions de chaque instance. Le résultat de LIME est un ensemble d'explications représentant la contribution de chaque attribut à une prédiction d'une seule instance, ce qui est une forme d'interprétabilité locale.

Dans LIME [RSG16], le modèle interprétable utilisé est une fonction linéaire, ou vecteur $\boldsymbol{w} \in \mathbb{R}^d$ dont le nombre de coefficients non nuls est minimisé. Ce modèle est entraîné sur de petites perturbations (échantillons de l'instance à expliquer) du modèle d'origine, telles que l'ajout de bruit, la suppression de mots ou le masquage de parties de l'image, etc. afin de fournir une bonne approximation locale des prédictions du modèle d'origine. En utilisant cette approche, LIME permet de mieux comprendre les raisons derrière les prédictions du modèle « boîte noire » et de fournir des explications interprétables qui aident à évaluer les décisions prises par le modèle d'apprentissage automatique.

Mathématiquement, les modèles de substitution locaux avec contrainte d'interprétabilité peuvent être exprimés comme suit :

$$\text{explication}(x) = \arg\min_{g \in G} \; L(f, g, \pi_x) + \Omega(g)$$

Le modèle d'explication pour l'instance x est le modèle g (par exemple, un modèle de régression linéaire) qui minimise la perte L (par exemple, l'erreur quadratique moyenne). Cette perte mesure la proximité de l'explication par rapport à la prédiction du modèle d'origine f (par exemple, un modèle XGBoost), tandis que la complexité du modèle $\Omega(g)$ est maintenue basse (par exemple, en minimisant dans une fonction linéaire le nombre d'attributs dont les valeurs sont différentes de zéro). G représente l'ensemble des explications possibles, par exemple tous les modèles de régression linéaire possibles. La mesure de proximité π_x définit la taille du voisinage autour de l'instance x que nous considérons pour l'explication. La méthode pour entraîner des modèles de substitution locaux est la suivante :

1. Sélectionnons l'instance cible pour laquelle nous souhaitons obtenir une explication de sa prédiction par un modèle « boîte noire ».
2. Perturbons ensuite cette instance et obtenons les prédictions du modèle pour les nouvelles instances obtenues.
3. Pondérons les nouveaux échantillons en fonction de leur proximité avec l'instance cible.
4. Entraînons un modèle interprétable sur l'ensemble de données obtenus.
5. Expliquons la prédiction en interprétant le modèle local.

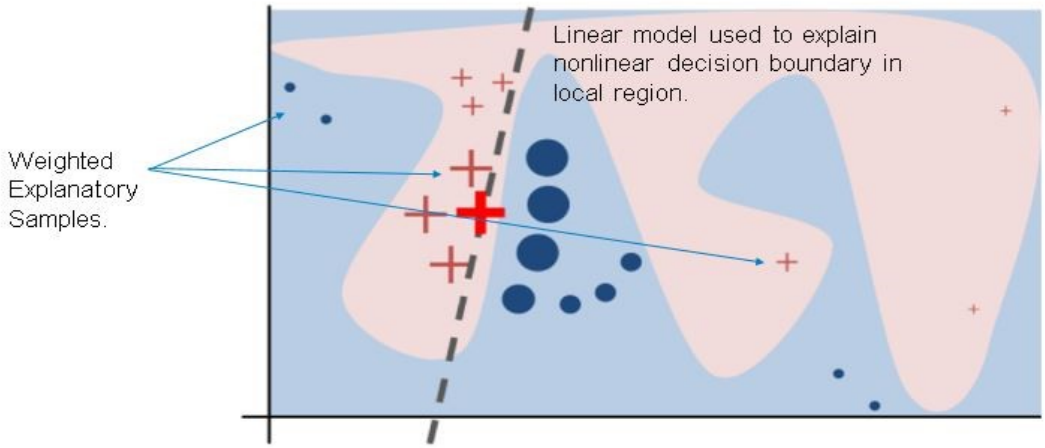


FIGURE 15 – LIME (source : [RSG16])

La figure 15 illustre le fonctionnement de LIME, la fonction de décision complexe du modèle de boîte noire f (inconnue de LIME) sur la figure 15 est représentée par l'arrière-plan bleu/rose, qui ne peut

pas être bien approché par un modèle linéaire en raison de sa structure complexe. La croix rouge en gras représente l'instance à expliquer. LIME échantillonne des instances autour du voisinage de l'instance à expliquer, et les pondère en fonction de leur distance avec l'instance à expliquer. La ligne en pointillés est l'explication apprise qui est localement (mais pas globalement) fidèle.

FIGURE 16 – Exemple d'une explication d'une mauvaise classification d'un husky « Husky vs Wolf » (source : [RSG16]).

Dans le but d'illustrer l'utilité de LIME, nous avons repris à la figure 16 l'exemple [RSG16], qui présente une classification entre les photos de loups et de chiens Eskimo (huskies). Un classifieur a été entraîné sur un ensemble d'images sélectionnées manuellement, de manière à ce que toutes les images de huskies aient de la neige en arrière-plan, tandis que les images de loups n'en avaient pas. Le classifieur prédit « Husky » s'il y a de la neige (ou un fond clair en bas), et « Loup » sinon, indépendamment de la couleur, de la position, de la posture, etc. de l'animal. La figure 16 montre l'explication fournie par LIME, représenté dans l'image de droite, montre que le modèle se base systématiquement sur la présence de neige pour classer chaque image comme étant un loup. Nous en concluons que le modèle n'a pas réellement appris à distinguer les loups des huskies, mais plutôt à reconnaître la présence ou l'absence de neige.

4.2.2 SHAP (*SHapley Additive exPlanations*)

Introduction. Une autre approche bien connue pour résoudre le problème de l'interprétabilité des boîtes noires est SHAP (*SHapley Additive exPlanations*). La notion de valeur de Shapley est issue du domaine de la théorie des jeux. Dans un jeu coopératif, les joueurs collaborent pour obtenir un certain gain. Le problème est alors la répartition de ce gain entre les différents acteurs. Shapley a proposé une méthode de répartition des gains d'une coalition de d joueurs. Sa solution s'applique au cas où l'utilité est transférable. La valeur de Shapley [Sha51] d'un attribut est sa contribution moyenne marginale dans toutes les coalitions possibles. Dans les jeux coopératifs, nous utilisons le concept de fonction caractéristique (noté g), qui donne la valeur $g(C)$ de la coalition C. Par exemple, si la coalition comprenant les joueurs 3 et 4 obtient un profit de 1200, nous écrivons $g(3, 4) = 1200$.

Nous pouvons décrire un jeu en indiquant les valeurs de la fonction caractéristique pour toutes les coalitions possibles, y compris celles ne comprenant qu'un seul joueur. Dans un jeu à d joueurs, il y a

$2^d - 1$ coalitions non vides. Par convention, la valeur de la fonction caractéristique d'une coalition vide est égale à 0. La valeur de Shapley pour le joueur i (noté ϕ_i(g)) étant donné une fonction caractéristique g est donnée par :

$$\phi_i(g) = \sum_{Z \subseteq N, i \in Z} \frac{(d - |Z|)!(|Z| - 1)!}{d!} (g(Z) - g(Z \backslash \{i\})) \tag{4}$$

où $N = [d]$ et Z est une coalition de joueurs.

La formule précédente exprime que pour obtenir $\phi_i(g)$, il est nécessaire de calculer la différence de gain $g(Z \cup i) - g(Z)$ pour chaque coalition Z dans laquelle le joueur i n'est pas présent. Cette différence de gain permet de comparer les résultats obtenus par la coalition avec et sans la participation du joueur i, ce qui permet d'évaluer l'impact de ce joueur lorsqu'il collabore avec Z. Si cette différence est positive, cela signifie que le joueur i apporte une contribution positive à la coalition. En revanche, si la différence est négative, cela indique que le joueur i pénalise le groupe. Si la différence est nulle, cela signifie que le joueur i n'apporte aucune valeur ajoutée à ce groupe. Ensuite, nous calculons la moyenne de ces différences pour toutes les coalitions dans auxquelles le joueur i participe.

Valeurs de Shapley en apprentissage automatique. Dans le contexte de l'apprentissage automatique, notre objectif est d'expliquer la prédiction d'un modèle $f(.)$ associée à une instance x. Chaque instance-prédiction est assimilée à un jeu : les joueurs sont les différentes valeurs x_i prises par les variables d'entrée. La quantité à répartir de manière équitable entre ces joueurs est la différence entre la prédiction $f(x)$ et la moyenne des prédictions $\mathbb{E}[f(x)]$. La fonction caractéristique du jeu est définie comme suit : $g(u) = \mathbb{E}[f(X)|X_u = x_u]$, où $X_u = x_u$ représente l'événement spécifiant les valeurs particulières x_u pour une coalition donnée u. Par exemple, si u correspond à la coalition $\{2, 3, 4\}$, alors $X_u = x_u$ indique l'événement $X_2 = x_2, X_3 = x_3, X_4 = x_4$.

SHAP. *Shapley Additive Explanation*, ou SHAP, est une approche de XAI proposée par [LL17] pour expliquer localement les prédictions d'un modèle d'apprentissage automatique en utilisant les valeurs de Shapley.

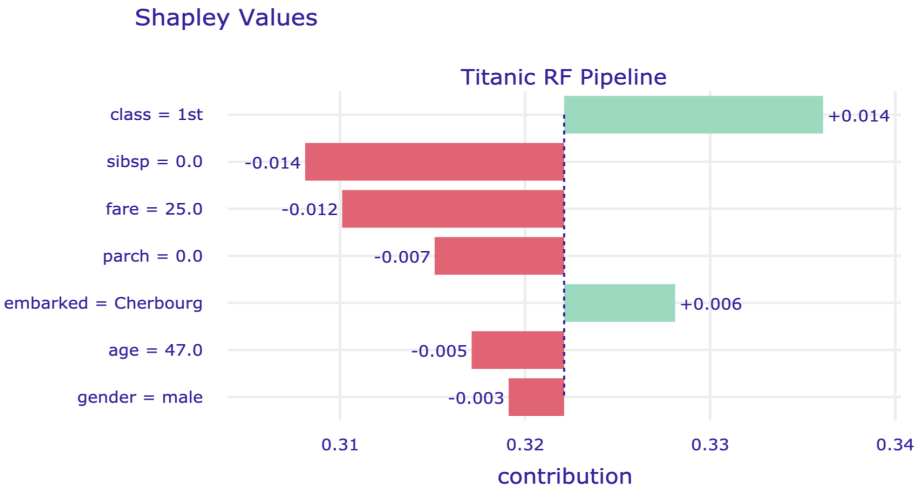


FIGURE 17 – Exemple pour illustrer SHAP (source : shap)

La figure 17 illustre l'intuition derrière SHAP. L'exemple est basé sur le célèbre jeu de données Titanic [7]. Dans cet exemple, une forêt aléatoire est utilisée pour effectuer une classification. L'instance cible est décrite dans la figure. L'instance est classée négative, et les barres rouges représentent les attributs qui ont contribué à cette classification négative (0), tandis que les barres bleues représentent les attributs qui ont contribué à la classification positive (1). Nous observons que la plupart des variables ont contribué à la classification négative. Les attributs qui ont eu le plus d'impact sur cette prédiction sont « *sibsp* » (le nombre de frères et sœurs/époux à bord du Titanic) et « *fare* » (le tarif du billet). En revanche, les variables « *age* » (l'âge du passager) et « *gender* » (le genre du passager) ont eu peu ou pas d'effet sur la prédiction.

KernelSHAP. En pratique, il est très difficile de calculer exactement les valeurs de Shapley en raison du grand nombre de coalitions possibles. Pour cette raison, les auteurs de [LL17] ont opté pour l'estimation de ces valeurs. Les valeurs de SHAP peuvent être approchées par échantillonnage [K14] ou, de manière équivalente, de la méthode d'influence quantitative des entrées [DSZ16]. Ces méthodes utilisent une approximation par échantillonnage d'une version des équations classiques des valeurs de Shapley. Des estimations d'échantillonnage distinctes sont effectuées pour chaque attribution. Bien qu'il soit raisonnable de les calculer pour un petit nombre d'entrées, la méthode Kernel SHAP proposée par [LL17] nécessite moins d'évaluations du modèle d'origine pour obtenir une précision d'approximation suffisamment bonne. En outre, des méthodes d'approximation spécifiques au modèle ont également été développées [LL17]. Ces méthodes sont :

— **DeepSHAP et GradientSHAP :** pour les réseaux de neurones.
— **LinearSHAP :** pour les modèles linéaires.

Au delà du fait que ces méthodes ne calculent pas des explications qui offrent des garanties de cohérence [HMS23], le calcul coûteux des valeurs de Shapley reste un inconvénient de la méthode.

TreeSHAP. Pour résoudre le problème du temps de calcul élevé des valeurs de Shapley, les auteurs de l'article [LEL18] ont proposé une approche spécifique aux modèles basés sur les arbres tels que les arbres de décision, XGBoost, LightGBM, Forêts aléatoires, etc. Cette approche permet de calculer rapidement et efficacement les valeurs de Shapley. De plus, ces valeurs sont exactes, localement précises. La complexité d'un algorithme standard appelé « Tree SHAP » pour le calcul des valeurs de Shapley dans les modèles basés sur les arbres est de l'ordre de $O(LT2^M)$, où T est le nombre d'arbres dans le modèle, L est le nombre maximal de feuilles dans un arbre et M est le nombre d'attributs. Dans l'article [LEL18], les auteurs proposent une version modifiée de cet algorithme qui enregistre le nombre de sous-ensembles S traversant chaque nœud de l'arbre. L'algorithme modifié a une complexité de calcul de $O(LTD^2)$, où D est la profondeur maximale de l'arbre. La réduction de complexité réalisée est très significative.

La figure 18 présente une comparaison des temps de calcul entre les méthodes KernelSHAP et TreeSHAP en fonction du nombre d'instances à expliquer. Dans l'expérience réalisée, un modèle de type Forêt aléatoire a été utilisé avec 25 arbres, et la base de données considérée est le jeu de données **« Compas »**. Il est clairement observé que le temps requis par la méthode KernelSHAP est significativement plus long que celui de la méthode TreeSHAP.

7. https ://www.kaggle.com/c/titanic/data

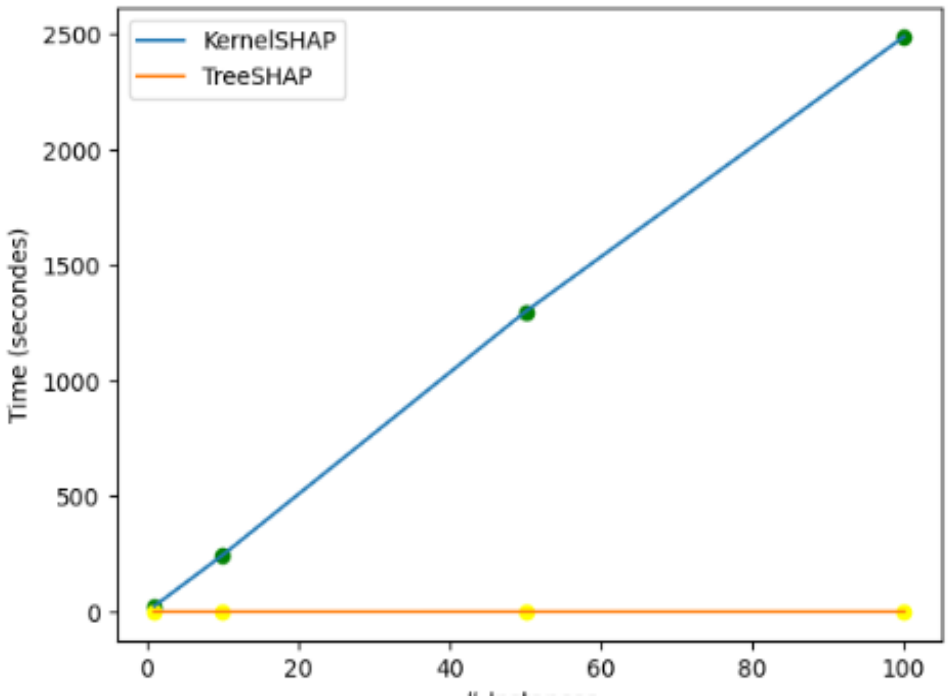


FIGURE 18 – Temps de calcul des valeurs de Shapley selon le nombre d'instances TreeSHAP vs KernelSHAP pour le *dataset* « compas ».

4.2.3 Anchors

Introduction. Nous présentons maintenant une approche agnostique qui explique les prédictions individuelles de tout modèle de classification de type « boîte noire » à l'aide de règles appelées Anchors (ou ancres) [RSG18]. Les auteurs de la méthode Anchors proposent un algorithme permettant de calculer efficacement ces règles pour n'importe quel modèle de type « boîte noire », avec des garanties en probabilité. Ils démontrent la flexibilité de l'approche en l'appliquant à une multitude de modèles différents, dans divers domaines et pour différentes tâches.

L'approche Anchors déploie une stratégie basée sur les perturbations pour générer des explications locales pour les prédictions des modèles d'apprentissage automatique de type « boîte noire ». Au lieu des modèles de substitution utilisés par LIME, les explications résultantes sont exprimées sous forme de règles logique faciles à comprendre. Ces règles sont réutilisables, car elles sont délimitées : les ancres incluent la notion de couverture, précisant à quelles autres instances, elles s'appliquent. À cette fin, des voisins, ou perturbations, sont créés et évalués pour chaque instance expliquée.

La figure 19 représente à la fois LIME et les ancres expliquant localement un classifieur binaire complexe (qui prédit « - » ou « + »). Les résultats de LIME n'indiquent pas à quel point ils sont fidèles, car LIME apprend uniquement une frontière de décision linéaire qui se rapproche le mieux du modèle étant donné un espace de perturbation.

Étant donné le même espace de perturbation, l'approche des ancres construit des explications dont la couverture est adaptée au comportement du modèle et l'approche exprime clairement leurs frontières. Ainsi, ils sont fidèles par conception et indiquent exactement pour quelles instances ils sont valides. Cette propriété rend les ancres particulièrement intuitives et faciles à comprendre.

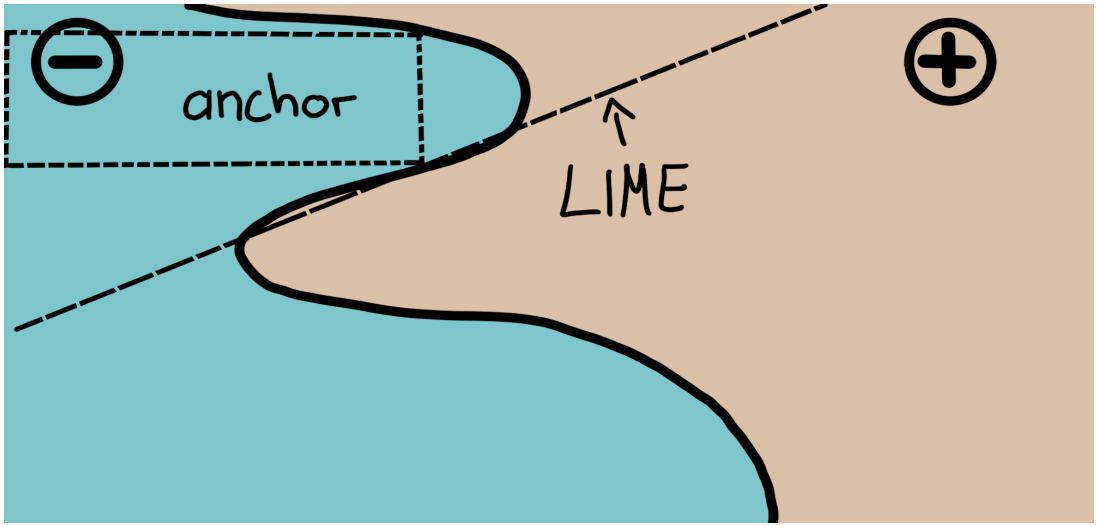


FIGURE 19 – LIME vs. Anchors. Une visualisation (source : anchor)

Formellement. Soit un modèle de type « boîte noire » $f : \mathcal{X} \to \mathcal{Y}$ et une instance $x \in \mathcal{X}$. La plupart des méthodes agnostiques fonctionnent en perturbant l'instance x selon une distribution de perturbation D_x (pour simplifier, D à partir de maintenant). Soit A une règle (ensemble de prédicats) agissant sur une telle représentation interprétable, de telle sorte que $A(x)$ renvoie 1 si sa condition est vrai pour l'instance x. Par exemple, dans la figure 19 (en haut), prenons $x =$ « Ce match n'est pas mal. », $f(x) =$ positif, $A(x) = 1$ où $A = \{$« pas », « mal »$\}$. Soit $D(\cdot|A)$ la distribution conditionnelle lorsque la règle A s'applique. A est une ancre si $A(x) = 1$ et A est une condition suffisante pour $f(x)$ avec une probabilité élevée, si un échantillon z provenant de $D(z|A)$ est susceptible d'être prédit comme positif (c-à-d $f(x) = f(z)$). Formellement, A est une ancre de précision τ si :

$$E_{D(z|A)}[1_{f(x)=f(z)}] \geq \tau, \;\; A(x) = 1.$$

Bien que la description mathématique des ancres puisse sembler claire et simple, la construction de règles spécifiques est souvent très difficile à réaliser. Les auteurs de l'article [RSG18] proposent une approche heuristique (car ils n'offrent aucune garantie), car il est souvent impossible d'évaluer toutes les instances de l'espace d'entrée. Cette approche heuristique est combinée avec la notion de couverture, qui mesure la probabilité d'une ancre. En maximisant à la fois la précision et la couverture, la procédure cherche à trouver la règle avec la plus grande couverture parmi toutes les règles éligibles, en prenant en compte le seuil de précision défini de manière probabiliste. Les règles avec des conditions plus fortes ont tendance à avoir une plus grande précision, mais elles peuvent être trop spécifiques et ne s'appliquer qu'à quelques instances, tandis que les règles avec des conditions moins fortes peuvent manquer de précision. Il existe donc un compromis entre précision et couverture.

Exemple. Prenons l'exemple du dataset « Compas » utilisé dans le domaine de la justice prédictive. Ce dataset contient des informations sur des individus arrêtés et prédit s'ils sont susceptibles de récidiver ou non. Supposons que nous ayons un modèle de prédiction basé sur l'apprentissage automatique qui utilise des attributs telles que l'âge, le genre, le type de délit, les antécédents criminels, etc. pour prédire la récidive de l'individu. En utilisant l'approche Anchors, nous pouvons extraire des règles explicatives qui déterminent les facteurs importants pour la prédiction de la récidive. Par exemple, si l'âge de l'individu est inférieur à 25 ans et qu'il a un historique de délits violents et des origines africaines, alors il est prédit comme *récidiviste* avec une précision de 90%.

4.2.4 Limites et faiblesses des approches agnostiques de l'état de l'art

Les approches agnostiques présentent l'avantage d'être applicables dans de nombreux cas pratiques où les modèles utilisés sont complexes. Cependant, la nature approximative des explications fournies fait qu'il est difficile d'être certain de leur validité dans le contexte du modèle d'origine. Les explications sont basées sur des heuristiques et des méthodes d'approximation, ce qui peut entraîner des biais et des imprécisions. Il est important d'être conscient de ces limitations lors de l'interprétation des résultats. Expliquer le raisonnement d'un modèle de « boîte noire » d'une manière générale à partir de sa structure interne est une tâche complexe et difficile. La recherche dans le domaine de l'XAI est donc justifiée, car il reste encore de nombreuses possibilités d'amélioration. Par exemple, la méthode LIME offre différentes possibilités d'approximation des boîtes noires, mais actuellement, seule une variante utilisant des fonctions linéaires est largement utilisée. Il y a donc un potentiel d'exploration et de développement de nouvelles approches d'approximation pour améliorer la compréhension des modèles d'apprentissage automatique. Nous rappelons pour conclure les principales faiblesses de LIME et SHAP.

LIME :
— Instances créées sans tenir compte des corrélations entre les variables.
— Cohérence des explications fournies (on peut avoir la même explication pour une instance classée positive et une autre classée négative [INM19]).
— Définition du «bon» voisinage pas claire.
— Méthode non soutenue par une vraie théorie mathématique.

SHAP :
— Temps de calcul très long (à l'exception de la méthode SHAP tree).
— Instabilité due à l'échantillonnage réalisé.
— Instances créées sans tenir compte des corrélations entre les variables.
— Cohérence des explications fournies (on peut avoir la même explication pour une instance classée positive et une autre classée négative).

Dans le domaine de l'XAI, il existe d'autres approches qui se basent sur des méthodes formelles offrant plus de garanties. Ces approches cherchent à ouvrir la boîte noire en exploitant la structure et les paramètres du modèle d'apprentissage automatique. En d'autres termes, elles tirent parti du fait que le modèle peut être un arbre de décision, une forêt aléatoire, un arbre boosté, etc. Ces approches permettent une meilleure compréhension et interprétation, car elles exploitent les structures spécifiques du modèle.

4.3 Méthodes formelles orientées modèles

Introduction. Cette section passe en revue les efforts de recherche développés depuis quelques années pour calculer des explications formelles rigoureuses des modèles d'IA. Nous passons ainsi en revue des méthodes qui nous permettent de calculer des explications formelles comme celles décrites dans [MS23, IMS21, Amg23, INAMS20b, ABB+22a, DH20]. Cette section se concentre sur les explications formelles et présente certaines de leurs caractéristiques.

L'idée centrale derrière les approches orientées modèles ou boîte blanche est d'exploiter l'architecture interne des classifieurs d'apprentissage automatique. Un courant de recherche majeur dans ce domaine vise à associer au classifieur considéré (qui est généralement considéré comme une boîte noire) une boîte blanche (ou transparente), représentée sous la forme d'une fonction booléenne ou d'un circuit

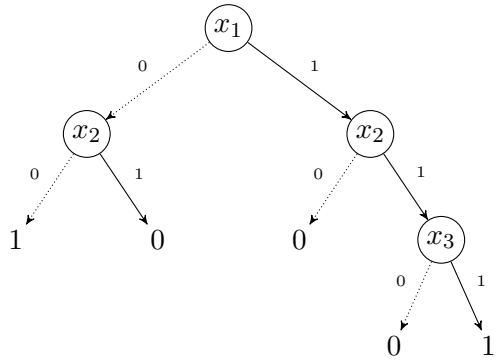


FIGURE 20 – Une représentation du classifieur h par un arbre de décision.

qui reproduit le comportement de la boîte noire en termes d'entrées et de sorties. Cela permet d'utiliser la boîte blanche pour répondre aux requêtes explicatives et expliquer les décisions prises par les classifieurs. La boîte blanche étant indépendante des entrées de la boîte noire, elle peut être pré-traitée (compilée) pour faciliter la génération d'explications sur les prédictions effectuées par la boîte noire et évaluer leur fiabilité.

Exemple 27. *Considérons le classifieur* $h : \{0,1\}^3 \rightarrow \{0,1\}$ *spécifié par la fonction polynomiale à seuil :* $h(\boldsymbol{x}) = 1 \Leftrightarrow x_1x_2x_3 + x_1x_2 - x_1 - x_2 \geq 0$. *Le classifieur* h *peut être représenté par un arbre de décision comme le montre la figure 20, ou par une formule* CNF *équivalente* ϕ_h *telle que :* $\phi_h = (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2 \wedge x_3)$.

Dans la suite, nous définissons certains types d'explications formelles proposées dans l'état de l'art, avec une attention particulière portée aux explications abductives et contrastives.

4.3.1 Explications abductives

Considérons un classifieur donné qui calcule une fonction de classification $\mathbf{C}$ sur l'espace des attributs X_d, une instance $\boldsymbol{x} \in X_d$, et soit t un sous-ensemble de l'ensemble des caractéristiques de $\boldsymbol{x}$, t est une explication abductive pour l'instance $\boldsymbol{x}$ étant donné $\mathbf{C}$ si pour toute instance $z \in X_d$ qui admet t pour sous-ensemble, on a $\mathbf{C}(\boldsymbol{z}) = \mathbf{C}(\boldsymbol{x})$. En d'autres termes, si les valeurs des caractéristiques dans t sont fixées conformément à t, la prédiction est garantie d'être $\mathbf{C}(\boldsymbol{x})$, indépendamment des valeurs attribuées aux autres attributs.

Définition 43 (explication abductive). *Pour un classifieur booléen* $\mathbf{C}$ *prenant en entrée une instance* $\boldsymbol{x} \in \{0,1\}^d$ *telle que* $\mathcal{C}(\boldsymbol{x}) = 1$*(resp.* $\mathcal{C}(\boldsymbol{x}) = 0$*), une* ***explication abductive*** *pour* $\boldsymbol{x}$ *étant donnée* $\mathbf{C}$ *(resp. de* $\neg\mathbf{C}$*) est un impliquant* t *de* $\mathbf{C}$ *qui couvre* $\boldsymbol{x}$.

Une explication abductive peut être considérée comme une réponse possible à la question pourquoi la prédiction du classifieur $\mathbf{C}$ est $\mathbf{C}(\boldsymbol{x})$. Dans ce mémoire, la représentation des explications à l'aide de sous-ensembles de caractéristiques sous forme de termes vise à simplifier leur compréhension. Une condition suffisante pour la prédiction est évidemment la conjonction de littéraux associés aux caractéristiques contenus dans l'explication abductive.

Exemple 28. *Considérons le classifieur* $h : \{0,1\}^3 \rightarrow \{0,1\}$ *spécifié par la fonction à seuil polynomiale :* $h(\boldsymbol{x}) = 1 \Leftrightarrow x_1x_2x_3 + x_1x_2 - x_1 - x_2 \geq 0$. *Pour l'instance* $\boldsymbol{x} = (1,1,1)$ *pour laquelle nous devons expliquer* $h(\boldsymbol{x}) = 1$, $x_1 \wedge x_2 \wedge x_3$ *est une explication abductive pour* $\boldsymbol{x}$ *étant donné* h.

4.3.2 Explications contrastives

Définissons maintenant des explications contrastives [Mil19, INAMS20a].

Définition 44 (explication contrastive). *Soit $\mathcal{C}$ un classifieur booléen et $\boldsymbol{x} \in \{0,1\}^d$ telles que $\mathcal{C}(\boldsymbol{x}) = 1$. Une* explication contrastive *pour $\boldsymbol{x}$ étant donnée $\mathcal{C}$ est un terme t sur X_d tel que $t \subseteq t_{\boldsymbol{x}}$ [8] et $t_{\boldsymbol{x}} \setminus t$ n'est pas un impliquant de $\mathcal{C}$.*

Une explication contrastive peut être considérée comme une réponse possible à la question pourquoi la prédiction du classifieur n'est pas une autre classe que $\mathcal{C}(\boldsymbol{x})$. Une autre perspective pour une explication contrastive est répondre à la question « comment ? », c'est-à-dire comment modifier les caractéristiques pour changer la prédiction.

Exemple 29. *Considérons encore une fois le classifieur $h : \{0,1\}^3 \to \{0,1\}$ spécifié par la fonction : $h(\boldsymbol{x}) = 1 \Leftrightarrow x_1x_2x_3 + x_1x_2 - x_1 - x_2 \geq 0$. Pour l'instance $\boldsymbol{x} = (1,1,1)$ pour laquelle nous avons $h(\boldsymbol{x}) = 1$, nous constatons que x_3 est une explication contrastive étant donné h et $\boldsymbol{x}$. En effet, il suffit de modifier la valeur de x_3 dans $\boldsymbol{x}$ pour changer la prédiction de 1 vers 0.*

Dualité entre les explications abductives et contrastives. Les travaux de [INMS19] soulignent des résultats de dualité entre explications abductives et explications contrastives quand on se focalise sur celles qui sont minimales pour l'inclusion. Les auteurs de [INAMS20a] ont exploré cette idée de dualité plus en détail. Cette correspondance permet de mieux comprendre comment les explications abductives et contrastives toutes ensembles peuvent offrir une vision plus complète de la justification des prédictions des modèles.

4.3.3 Notes complémentaires

Plusieurs méthodes ont été développées pour transformer les classifieurs d'apprentissage automatique de différents types en circuits booléens, tels que des fonctions booléennes ou des formules en formes normales, ayant les mêmes comportements en termes d'entrée et de sortie que les classifieurs d'apprentissage automatique. Grâce à ces transformations, les requêtes d'explicabilité de l'IA (XAI) relatives aux classifieurs peuvent être déléguées aux circuits correspondants. Dans cette section, nous présentons quelques requêtes de vérification concernant les classifieurs [AKM20].

Mesurer la fréquence d'un certain attribut dans une classe donnée (MFR). **MFR** a pour objectif d'évaluer l'importance de la présence d'une caractéristique x_k dans chaque classe d'intérêt. Plus généralement, il est possible de considérer une combinaison de caractéristiques (chaque caractéristique étant présent ou absente) au lieu d'une seule caractéristique présente. De même, il est possible de prendre en compte une combinaison de classes (chaque classe étant présente ou non). Lorsque la fréquence est égale à 1, la présence de la caractéristique x_k est nécessaire pour que l'instance soit reconnue faisant partie de la classe y_j, tandis que lorsque la fréquence est égale à 0, la présence de x_k est interdite pour que l'instance soit reconnue comme faisant partie de la classe y_j.

Identification d'attributs non-pertinents pour une classe donnée (IIR). Dans de nombreux scénarios, l'utilisateur a des connaissances quant aux attributs qui devraient être pertinents (ou non) pour certaines classes. Par exemple, l'utilisateur peut s'attendre à ce que la forme d'un fruit soit pertinente

8. Nous rappelons qu'étant donné une instance $\boldsymbol{a} \in \{0,1\}^d$, $t_{\boldsymbol{a}}$ est l'ensemble des littéraux sur $X_d = \{x_1, \cdots, x_d\}$ correspondant à l'instance $\boldsymbol{a}$. En d'autres termes, pour tout $i \in [d]$, le littéral x_i (resp. $\overline{x_i}$) appartient à $t_{\boldsymbol{a}}$ si et seulement si la i-ème coordonnée de $\boldsymbol{a}$ vaut 1 (resp. 0). Le terme correspondant à l'instance $\boldsymbol{a}$ est : $\bigwedge_{i=1}^{d} x_i^{a_i}$ où $x_i^0 = \overline{x}_i$ et $x_i^1 = x_i$.

pour déterminer s'il s'agit d'une banane, ce qui est effectivement le cas dans cet exemple. À l'inverse, l'utilisateur pourrait estimer qu'une classe donnée ne devrait pas dépendre de certaines attributs, car cela révélerait un biais dans le classifieur.

Identification d'attributs monotones/anti-monotones (IMO). Dans de nombreuses applications, il est courant de considérer que l'augmentation de la valeur d'un attribut n'affecte pas l'appartenance à une classe donnée. De même, on peut également s'attendre à ce que la diminution de la valeur d'un autre attribut n'affecte pas non plus cette appartenance. Par exemple, si un fruit est initialement identifié comme une pomme en raison de sa couleur non verte, le fait de changer sa couleur en vert ne devrait pas changer le fait qu'il s'agit toujours d'une pomme. En revanche, si un fruit est initialement reconnu comme un raisin en raison de son goût sucré, on peut raisonnablement s'attendre à ce qu'il reste un raisin même si son goût devient acide (c'est-à-dire non sucré).

4.4 Conclusion

Dans ce chapitre, nous avons présenté un panorama des explications dans le domaine de l'intelligence artificielle explicable. Nous avons abordé différentes formes d'explications, en mettant en évidence la variabilité des formes et les propriétés liées à la capacité de raisonner à partir des explications. Nous avons également présenté deux types majeurs de méthodes pour générer des explications. Les méthodes agnostiques locales, telles que LIME, SHAP (*SHapley Additive exPlanations*) et Anchors, sont utiles et efficaces pour les boîtes noires, notamment pour les réseaux de neurones, en fournissant des explications locales compréhensibles. Cependant, il convient de noter leurs faiblesses et leurs limites, en particulier l'instabilité et l'incohérence des explications calculées avec ces méthodes. Cela les rend peu fiables pour être utilisées, notamment dans des domaines sensibles tels que la médecine, la sécurité et la finance. Il est donc essentiel de développer des méthodes d'explication plus robustes, pour garantir des explications de qualité, transparentes et dignes de confiance.

Les explications formelles orientées modèles, telles que les explications abductives et les explications contrastives, ont été utilisées pour des boîtes blanches, où la structure interne du modèle est connue. Ces méthodes fournissent des explications basées sur le raisonnement logique. Bien que ces approches présentent de nombreux avantages, il convient également de noter qu'elles ont leurs limites et leurs faiblesses. Elles nécessitent une connaissance préalable du modèle, ce qui peut limiter leur applicabilité dans des scénarios où les détails du modèle ne sont pas disponibles. L'explicabilité formelle s'appuie largement sur plusieurs domaines de recherche en IA, notamment le raisonnement automatisé. Différents solveurs SAT et MaxSAT peuvent être exploités pour trouver des moyens de calculer des explications, à la fois exactes et probabilistes, ainsi que pour répondre aux requêtes d'explicabilité.

En conclusion, l'étude des méthodes d'explication continue d'évoluer pour répondre aux défis de l'interprétabilité des modèles d'apprentissage automatique. Il est important de poursuivre la recherche dans le domaine de l'intelligence artificielle explicable afin de développer des approches plus complètes et adaptées à divers types de modèles et de problèmes, pour fournir des explications de haute qualité, précises et fiables, tout en répondant aux exigences de transparence et de confiance dans les systèmes d'IA.

Conclusion de la partie « état de l'art »

Dans cette étude de l'état de l'art, nous avons exploré différents aspects de l'explicabilité dans le domaine de l'intelligence artificielle. Nous avons mis en évidence la croissance de l'apprentissage automatique et l'importance de l'explicabilité en tant que facteur déterminant. Nous avons également présenté les concepts d'explicabilité et d'interprétabilité, ainsi que différents types d'explications et les méthodes associées.

Nous avons aussi présenté des concepts issus de divers domaines clés qui nous sont utiles dans la suite de ce mémoire, tels que la logique propositionnelle et le problème SAT, MaxSAT et le problème de comptage de modèles, la complexité algorithmique, l'optimisation combinatoire à travers la sous-modularité et la super-modularité. Pour chacun de ces domaines nous avons mis en évidence les concepts clés, les définitions et les méthodes associées dont nous avons besoin dans la partie « contribution » de ce mémoire.

Nous avons également exploré différentes formes d'explication, ainsi que les propriétés liées à la capacité de raisonner à partir d'explications. Nous avons examiné les méthodes permettant de dériver des explications, notamment les méthodes agnostiques locales telles que LIME, SHAP et Anchors. Nous avons identifié leurs limites et leurs faiblesses.

Enfin, nous avons abordé le monde des explications formelles et des méthodes de calcul d'explications formelles. Ces méthodes dépendent généralement de l'architecture interne du modèle. Nous avons rappelé les notions d'explication abductive et d'explication contrastive, qui sont largement utilisées pour expliquer les boîtes blanches où la structure interne du modèle est connue. Ces méthodes fournissent des explications robustes, fiables et de haute qualité. Cependant, elles nécessitent une connaissance préalable du modèle, ce qui limite leur applicabilité dans certains scénarios. Elles sont aussi souvent difficiles à calculer, induisant un problème de passage à l'échelle.

En conclusion, cette étude a mis en évidence l'importance de l'explicabilité dans le domaine de l'intelligence artificielle et a présenté différentes approches et méthodes pour générer des explications. Il est crucial de poursuivre la recherche dans ce domaine afin de développer des approches, répondant aux exigences de précision, de généralisation et de transparence. L'exploitation des connaissances en logique propositionnelle, en optimisation combinatoire et en apprentissage automatique peut contribuer à améliorer l'explicabilité des systèmes d'IA et à renforcer la confiance dans leurs décisions.

Deuxième partie

Contributions

Introduction

Dans cette partie, nous présentons les contributions que nous avons apportées au cours de cette thèse. Cette partie est structurée de la manière suivante. Le premier chapitre (chapitre 5) explore le concept d'**intelligibilité computationnelle** des classifieurs booléens, caractérisée par leur aptitude à répondre aux requêtes de XAI en temps polynomial [ABB+21]. Nous examinons divers types de classifieurs, tels que les arbres de décision, les formules normales disjonctives (DNF), les listes de décision, les règles de décision, les ensembles d'arbres et les réseaux neuronaux booléens. S'appuyant sur des requêtes de XAI, englobant à la fois des requêtes d'explication et de vérification, notre étude révèle un **écart significatif d'intelligibilité entre les différentes familles de classifieurs**. Les arbres de décision permettant de traiter efficacement ces requêtes, tandis que les formules DNF, les listes de décision, les forêts aléatoires, les arbres boostés, les perceptrons multicouches booléens et les réseaux neuronaux binaires éprouvent des difficultés à traiter efficacement les différentes requêtes.

Le chapitre 6 se consacre à l'examen de la pertinence des arbres de décision en tant que modèles privilégiés dans des domaines sensibles où l'interprétabilité est primordiale. Cette section met en évidence la capacité explicative distinctive des arbres de décision, en soulignant leur transparence intrinsèque. De plus, elle explore les explications qu'ils offrent. Nous nous focalisons sur les notions d'explications abductives, notamment les raisons suffisantes et les raisons suffisantes de taille minimale. Nous introduisons également les notions d'attributs pertinents et nécessaires, ainsi que l'importance explicative d'un attribut. Nous démontrons que le nombre des raisons suffisantes minimales pour une instance donnée peut être exponentiel dans le nombre d'attributs, rendant leur génération compliquée, ainsi que leur exploitation par l'utilisateur. Pour pallier ce problème, nous proposons d'exploiter des préférences utilisateur, et de calculer uniquement des explications préférées. Au chapitre 7, plusieurs modèles de préférences sont présentés et discutés. Pour chaque modèle, nous présentons et évaluons un algorithme pour calculer des raisons suffisantes préférées pour les arbres de décision. Nous montrons que, en pratique, les raisons suffisantes préférées pour une instance peuvent être beaucoup moins nombreuses que ses raisons suffisantes.

Les raisons suffisantes sont parfois de trop grande taille pour être interprétables. Dans ces cas, nous devons réduire la taille des raisons suffisantes tout en déterminant toujours l'étiquette prédite avec une probabilité élevée. Au chapitre 8, nous montrons que calculer de telles explications probabilistes est NP-difficile, même pour les arbres de décision. Afin de contourner ce problème, nous examinons l'approximabilité des explications probabilistes à travers le prisme de la super-modularité. Nous étudions à la fois les approches de descente gloutonne et d'ascension gloutonne pour la minimisation super-modulaire, dont les garanties d'approximation dépendent de la courbure de la fonction d'erreur « non normalisée » qui évalue la précision de l'explication. Basés sur diverses expériences pour expliquer les prédictions des arbres de décision, nous montrons que nos algorithmes gloutons offrent une alternative efficace aux algorithmes de calcul exact d'explications abductives.

5

Sur l'intelligibilité computationnelle des classifieurs booléens

Sommaire

5.1 Requêtes XAI 81
5.1.1 Introduction 81
5.1.2 Requêtes de vérification 82
5.1.3 Requêtes d'explication 83
5.1.4 Perceptrons multi-couches booléens « PMC » et réseaux de neurones binaires « BNN » 84
5.2 Complexité des requêtes XAI pour les classifieurs booléens 86
5.2.1 Le cas des requêtes de vérification 87
5.2.2 Le cas des requêtes d'explication 89
5.3 Sur l'intelligibilité des classifieurs booléens 90
5.3.1 Sur l'intelligibilité des requêtes XAI pour les arbres de décision 91
5.3.2 Sur l'intelligibilité des requêtes XAI pour les classifieurs booléens 94
5.4 Discussion 99
5.5 Conclusion 100

Dans ce chapitre, nous examinons l'*intelligibilité computationnelle* des classifieurs booléens, ce qui se traduit par leur capacité à répondre aux requêtes d'IA explicative en un temps polynomial. Les familles de classifieurs que nous considérons incluent les arbres de décision, les formules DNF, les listes de décision, les règles de décision, les ensembles d'arbres et les réseaux de neurones binaires (BNN). En utilisant un ensemble de requêtes XAI, englobant à la fois des requêtes d'explication et de vérification, nous démontrons l'existence d'un *écart significatif en termes d'intelligibilité entre ces différentes familles de classifieurs*. D'un côté, toutes les requêtes de XAI peuvent être gérées efficacement avec les arbres de décision. D'un autre côté, sauf si P = NP, aucune de ces requêtes ne peut être traitée en temps polynomial pour les formules DNF, les listes de décision, les forêts aléatoires, les GBT, les perceptrons multi-couches booléens, ou encore les réseaux de neurones binaires.

Ces travaux ont donné lieu à la publication [ABB+21].

5.1 Requêtes XAI

5.1.1 Introduction

Qu'est-ce qu'un bon classifieur ? Répondre à cette question nécessite l'identification de plusieurs critères afin d'évaluer la qualité des classifieurs. Un critère clé pour mesurer la capacité de généralisation des classifieurs est *la précision*. Étant donné une distribution de probabilité sur les données, la précision d'un classifieur (booléen) est définie par la probabilité d'étiquetage correct d'une instance de données aléatoire. En apprentissage statistique, la distribution de probabilité est inconnue et nous n'avons accès qu'à un échantillon de données pour apprendre le classifieur. Le problème d'apprentissage est ainsi formulé comme une tâche d'optimisation stochastique : étant donné une famille de classifieurs candidats, souvent appelée classe de concepts, trouver un classifieur dans la famille qui minimise l'erreur empirique (éventuellement régularisée) sur l'échantillon d'entraînement. En pratique, la précision de la classification est estimée sur des échantillons de test à l'aide de mesures d'évaluation telles que la validation croisée. Cependant, la précision n'est pas le seul critère pour choisir un classifieur : dans de nombreuses applications du monde réel, un autre critère important est *l'intelligibilité*. En termes généraux, un classifieur est intelligible si ses prédictions peuvent être *expliquées* de manière compréhensible à un utilisateur et si son comportement peut être *vérifié* comme conforme aux attentes de l'utilisateur. L'exigence d'explicabilité est un problème juridique en Europe depuis la mise en œuvre du Règlement général sur la protection des données (RGPD) de l'Union Européenne 2016/679, le 25 mai 2018 [GF17]. En conséquence, l'intelligence artificielle explicative (XAI) est devenue un sujet de recherche très actif ces dernières années (voir par exemple [BTT+18, INM19, CZS+19, CZZvdS20, JR20]).

Malgré l'importance des deux critères, l'intelligibilité semble être beaucoup plus difficile à circonscrire que la précision des classifieurs. En effet, dans la littérature sur l'apprentissage statistique, la capacité de généralisation des classifieurs a été caractérisée formellement à travers les prismes de la capacité d'apprentissage [Val84, Hau92], de convergence uniforme et de la stabilité algorithmique [SSSSS10, CP18]. En revanche, le terme « intelligibilité » ne possède pas de signification commune largement partagée, car il dépend de divers desiderata visant à clarifier le comportement du classifieur dans certaines situations pratiques [Lip18]. Cependant, différentes *formes* d'intelligibilité peuvent être formalisées, en mettant l'accent sur la capacité du classifieur à répondre correctement à certains questions. De telles formes d'intelligibilité se reflètent dans les *requêtes d'explication et de vérification* introduites dans la littérature XAI. Notamment, un classifieur devrait être équipé de *facilités d'explication*, comprenant, par exemple, la capacité d'identifier des attributs pertinents qui, ensemble, sont suffisants pour prédire la classe d'une instance de données. De plus, le classifieur devrait être *accessible à l'inspection*, surtout lorsque l'utilisateur a des attentes concernant le comportement du classifieur, et qu'il souhaite vérifier si le classifieur est conforme à ses attentes. Par exemple, dans un problème de classification de prêts, si un prêt est accordé à un client qui n'a pas un revenu élevé, alors le prêt ne devrait pas être refusé lorsque le revenu augmente, en admettant que les autres attributs restent inchangés. Cette attente peut être formalisée à l'aide d'une requête de vérification qui vérifie le comportement monotone du classifieur par rapport à l'attribut lié au revenu du client.

Répondre à de telles requêtes XAI nécessite la disponibilité d'algorithmes d'inférence pour calculer des réponses dans un délai raisonnable. De ce point de vue, chaque requête peut être considérée comme une propriété qu'une famille de classifieurs peut offrir ou non : elle est offerte lorsqu'il existe un algorithme en temps polynomial pour répondre à la requête à partir de n'importe quel classifieur de la méme famille, et elle ne l'est pas lorsqu'il n'existe pas un tel algorithme, sauf si $\mathsf{P} = \mathsf{NP}$. En d'autres termes, l'*intelligibilité computationnelle* d'une famille de classifieurs peut être définie comme l'ensemble des re-

Requête XAI	Description
EMC	Énumération des explications de cardinalité minimale
DPI	Calcul d'un impliquant premier
ECO	Énumération des explications contrefactuelles
CIN	Comptage des instances associées à une classe donnée
EIN	Énumération des instances associées à une classe donnée
IMA	Identification des attributs obligatoires ou interdits
IIR	Identification des attributs non pertinents pour une classe donnée
IMO	Identification des attributs monotones / anti-monotones
MCP	Mesure de la proximité d'une classe par rapport à un prototype

TABLE 4 – Requêtes XAI

quêtes XAI réalisables par la famille.[9] Notre objectif dans ce chapitre est d'ouvrir une voie vers l'étude l'intelligibilité computationnelle des classifieurs booléens. Comme point référence, nous utilisons des requêtes d'XAI parmi celles considérées dans [AKM20], qui sont résumées dans le tableau 4 : **EMC, DPI et ECO** sont des requêtes d'explication, et **CIN, EIN, IMA, IIR, IMO, MCP** sont des requêtes de vérification.

Étant donné ces requêtes XAI, nous examinons 7 familles de classifieurs booléens : les arbres de décision, les formules DNF, les listes de décision, les forêts aléatoires, les arbres boostés (GBT), les perceptrons multicouches booléens et les réseaux de neurones binaires. La principale contribution de ce travail réside dans l'identification de résultats de complexité établissant l'existence d'un *important écart d'intelligibilité entre les familles de classifieurs*, estimé par le nombre de requêtes XAI qui sont réalisables. Plus précisément, nous démontrons que toutes les requêtes XAI sont réalisables pour la famille des arbres de décision, tandis qu'aucune d'entre elles n'est réalisable en temps polynomial pour aucune des autres familles citées (sauf si $\mathsf{P} = \mathsf{NP}$).

5.1.2 Requêtes de vérification

Les requêtes XAI prises en considération dans ce chapitre sont divisées en requêtes de vérification et en requêtes d'explication. Ces requêtes sont proposées dans [AKM20] et sont définies pour la classification multi-étiquettes (le classifieur $\mathbf{C}$ associe à chaque instance un vecteur $\mathbf{c}$ d'étiquettes binaires). Nous rappelons d'abord quelques requêtes de vérification.

Comptage des entrées (CIN) / énumération des entrées (EIN) associé à une classe donnée. Ces interrogations sont pertinentes pour un utilisateur qui cherche à saisir les catégories telles qu'elles sont identifiées par le classifieur $\mathbf{C}$ (ce qui peut différer des attentes initiales de l'utilisateur). Par exemple, un utilisateur pourrait être surpris de constater qu'une instance spécifique appartient à une classe différente de celle qu'il avait espérée ou que 90% des instances sont positives alors qu'il pense qu'il n'y en a pas plus que 60%.

Identification des attributs obligatoires ou interdits (IMA). Cette requête vise à évaluer la pertinence de la présence de chaque attribut x_i dans chaque classe cible c_j. Plus généralement, lorsque la

9. Cette approche rappelle l'évaluation computationnelle des langages de *représentation des connaissances* réalisée dans le cadre de la compilation des connaissances [DM02].

fréquence de présence de x_i est égale à 1, x_i est nécessaire pour être identifié comme un élément de c_j, tandis que lorsqu'elle est égale à 0, la présence de x_i est interdite pour que l'instance soit reconnue comme un élément de c_j.

Identification des attributs non pertinents pour une classe donnée (IIR). Dans de nombreux scénarios, l'utilisateur a certaines croyances concernant les attributs qui doivent être pertinents (ou non) pour certaines classes. Par exemple, il pourrait s'attendre à ce que la forme d'un fruit soit pertinente pour déterminer s'il s'agit d'une banane. A contrario, l'utilisateur pourrait croire qu'une classe donnée ne devrait pas dépendre de certains attributs, ce qui révélerait autrement un biais dans le classifieur. Par exemple, il pourrait s'attendre à ce que la classe des bananes ne dépende pas uniquement de sa forme. Rappelons qu'un attribut d'entrée x_i est sans importance pour une classe c_j si la valeur de x_i peut être modifiée sans changer le fait que l'instance est prédite par le classifieur comme étant un élément de c_j. En d'autres termes, l'appartenance à c_j de toute instance $\boldsymbol{x}$ ne dépend pas de la valeur de son attribut x_i.

Identification des attributs monotones / anti-monotones (IMO). Dans de nombreuses applications, on estime que l'augmentation de la valeur d'un attribut ne modifie pas l'appartenance à une classe donnée. On peut également s'attendre à ce que la diminution de la valeur d'un autre attribut ne modifie pas non plus l'appartenance à cette classe. Par exemple, si un fruit donné de couleur non verte est reconnu comme une pomme, alors le fait de changer sa couleur en verte ne devrait pas modifier le fait qu'il reste une pomme. En revanche, si un fruit au goût sucré est reconnu comme du raisin, il est raisonnable de penser qu'il resterait du raisin même si son goût était acide (c'est-à-dire non sucré).

Mesurer la proximité d'une classe par rapport à un prototype (MCP). L'utilisateur peut aussi avoir en tête un élément prototype de la classe. Par exemple, il peut croire qu'une pomme a une forme ronde, n'est pas verte et a un goût sucré, ce qui conduit a un prototype. En se basant sur ces informations de base, il est intéressant de déterminer dans quelle mesure chaque classe (telle qu'elle est reconnue par $\mathbf{C}$) correspond à ce prototype, ce qui peut être évalué en calculant la *distance de Hamming* entre chaque élément de la classe et le prototype, et en considérant la distance maximale. Une grande valeur de cette distance peut être considérée comme inacceptable, car elle peut refléter un biais dans les données utilisées pour former $\mathbf{C}$.

5.1.3 Requêtes d'explication

Nous nous tournons maintenant vers les requêtes d'explication, qui sont liées à une entrée spécifique $\boldsymbol{x}$ et à la manière dont elle a été classée par $\mathbf{C}$. Souvent, la raison pour laquelle $\boldsymbol{x}$ a été classée comme un élément d'une classe c_j ne dépend pas de tous les attributs de $\boldsymbol{x}$, mais seulement de certains d'entre eux. Cela conduit aux deux notions suivantes d'explications de cardinalité minimale et d'explications d'impliquant premier introduites dans [SCD18a]. Lorsque la prédiction obtenue par $\mathbf{C}$ est quelque peu inattendue, l'utilisateur peut également demander des explications contrefactuelles. Quel que soit le type d'explication recherchée (de cardinalité minimale, d'impliquant premier, contrefactuelle), une instance peut avoir un nombre exponentiel d'explications.

Énumération des explications de cardinalité minimale (EMC). Étant donné une entrée $\boldsymbol{x}$ telle que $C(\boldsymbol{x}) = \boldsymbol{c}$ avec $c_j = 1$, une explication de cardinalité minimale de $\boldsymbol{x}$ classée comme c_j est une instance d'entrée $\boldsymbol{x}'$ telle que $C(\boldsymbol{x}') = \boldsymbol{c}'$ avec $c'_j = 1$ (c'est-à-dire que $\boldsymbol{x}'$ est également classée comme c_j), $\boldsymbol{x}'$ est cohérente avec $\boldsymbol{x}$ sur les uns, au sens que pour tout $k \in [d]$, si $x'_k = 1$, alors $x_k = 1$, et $\boldsymbol{x}'$ a un nombre

minimal de coordonnées définies à 1. Ainsi, les caractéristiques définies à 1 dans $\boldsymbol{x}'$ sont suffisantes pour expliquer pourquoi $\boldsymbol{x}$ a été classé par C en tant qu'élément de c_j.

Calcul d'un impliquant premier (DPI). Étant donné une entrée $\boldsymbol{x}$ telle que $\mathbf{C}(\boldsymbol{x}) = c$, une explication d'impliquant premier est sous ensemble t minimal pour l'inclusion des caractéristiques de $\boldsymbol{x}$ tel que toute instance $\boldsymbol{x}'$ couverte par t vérifie $\mathbf{C}(\boldsymbol{x}')$. Les caractéristiques de t peuvent être interprétées comme expliquant pourquoi $\boldsymbol{x}$ a été classé comme c par le classifieur $\mathbf{C}$.

Énumération des explications contrefactuelles (ECO). Parfois, l'utilisateur peut être surpris par la manière dont $\mathbf{C}$ a classé une instance donnée $\boldsymbol{x}_1$. Elle s'attendait à ce que l'instance d'entrée soit reconnue comme une pomme, mais elle a été classée comme du raisin (c_j). Dans un tel cas, l'utilisateur peut demander une explication contrefactuelle : on cherche à identifier une instance $\boldsymbol{x}_2$ qui est classée comme une pomme (et non comme du raisin), et qui soit aussi proche que possible de $\boldsymbol{x}_1$ en termes de nombre de valeurs communes des attributs dans les deux entrées (autrement dit, pour la distance de Hamming $d_H(\boldsymbol{x}_1, \boldsymbol{x}_2)$). En effet, l'ensemble des attributs qui diffèrent entre $\boldsymbol{x}_1$ et $\boldsymbol{x}_2$ peut être considéré comme une explication du pourquoi $\boldsymbol{x}_1$ n'a pas été classé comme $\boldsymbol{x}_2$.

5.1.4 Perceptrons multi-couches booléens « PMC » et réseaux de neurones binaires « BNN »

Dans cette section, nous allons d'abord brièvement rappeler quelques notions préliminaires qui sont pertinentes pour ce chapitre. De plus, nous allons rappeler les définitions formelles de deux familles de classifieurs : les perceptrons multi-couches booléens et les réseaux de neurones binaires.

Préliminaires. Nous rappelons que pour un entier positif d, $[d]$ est l'ensemble $\{1, \ldots, d\}$. Soit $\mathcal{F}_d$ l'ensemble de toutes les fonctions booléennes de $\mathbb{B}^d$ dans $\mathbb{B}$, où $\mathbb{B} = \{0, 1\}$. Tout vecteur $\boldsymbol{x}$ dans l'hypercube booléen $\mathbb{B}^d$ est appelé une *instance*, $\boldsymbol{x}$ est un exemple positif (ou *modèle*) d'une fonction f si $f(\boldsymbol{x}) = 1$, et $\boldsymbol{x}$ est un exemple négatif de f si $f(\boldsymbol{x}) = 0$. Dans ce qui suit, nous utilisons $\top$ et $\bot$ pour désigner respectivement $\top(\boldsymbol{x}) = 1$ et $\bot(\boldsymbol{x}) = 0$ pour tout $\boldsymbol{x} \in \mathbb{B}^d$.

Perceptrons multi-couches booléens. Un réseau de neurones est formé à partir de sommets d'un graphe orienté, avec les arcs du graphe décrivant les flux des signaux. Plus formellement, un réseau de neurones *feedforward* est défini par un graphe acyclique orienté (V, E), et une fonction de poids sur les arcs $w : E \to \mathbb{R}$. Chaque nœud $v \in V$ du graphe correspond à un neurone. Dans un réseau de neurones *multi-couches*, l'ensemble des nœuds est décomposé en une union de sous-ensembles (non vides) disjoints $V = \bigcup_{l=1}^{q} V_l$, de telle sorte que chaque arc dans E relie chaque nœud dans V_l à chaque nœud dans V_{l+1}, pour un certain $l \in [q-1]$. En conséquence, chaque neurone $v \in V$ correspond à une paire l, i où $l \in [q]$ est une couche, et $i \in [|V_l|]$ est un rang dans la couche l. La couche inférieure V_1 est appelée la couche d'entrée et contient n sommets. Les couches $V_2, \cdots, V_{q-1}$ sont appelées couches *cachées*, et la couche supérieure V_q est appelée la couche de sortie. Les entrées du $i-$ème neurone de la couche l avec $1 < l \leq q$ sont les sorties de tous les neurones de la couche $l-1$. Le i-ème neurone de k couche l possède plus une entrée additionnelle $b_{l,i} \in \mathbb{R}$, appelée biais. Nous notons par $o_{l,i}(\boldsymbol{x})$ la sortie du $i-$ème neurone de la couche l lorsque le réseau prend en entrée l'instance $\boldsymbol{x} \in \mathbb{R}^d$. Avec cette notation en main, un réseau neuronal multi-couches peut être spécifié de manière récursive comme suit :

$$o_{1,i}(\boldsymbol{x}) = x_i$$

$$o_{l,i}(\boldsymbol{x}) = sgn\left(\sum_{j:(v_{l-1,j}, v_{l,i}) \in E} w(v_{l-1,j}, v_{t,i}) o_{l-1,j}(\boldsymbol{x}) + b_{l,i}\right)$$

où sgn est la fonction de signe telle que $sgn(z) = \mathbb{1}[z \geq 0]$. La profondeur, la largeur et la taille du réseau de neurones est donnée par q, $\max_l |V_l|$ et $|V|$ respectivement. Dans un *perceptron multi-couches booléen*, également connu sous le nom de *réseau à seuil multi-couches booléen* P [Ant01], les instances d'entrée sont des vecteurs dans l'hypercube $\mathbb{B}^d$, et la couche de sortie se compose d'un seul neurone pour lequel la sortie, notée $P(\boldsymbol{x})$, est une valeur booléenne dans $\mathbb{B}$.

Exemple 30. *Un exemple d'un perceptron multi-couches booléen P est donné à la figure 21. Le poids de chaque arc est attaché sous forme d'étiquette à l'arc correspondant. Le biais associé à chaque neurone dans la couche 2 est -1, et le biais associé au neurone unique dans la couche 3 est -3. Le biais associé à un neurone est donné dans la boîte représentant le neurone dans la figure.*

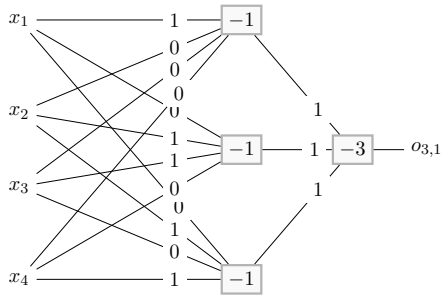


FIGURE 21 – Un exemple de perceptron multi-couches booléen

Réseau de neurones binaires « BNN ». Introduits dans [HCS+16], les *réseaux de neurones binaires* sont des réseaux de neurones multi-couches dont les activations et les poids sont principalement binaires (mais variant dans $\{-1, 1\}$). Un **BNN** est généralement décrit en termes de composition de q blocs de couches (assemblés séquentiellement) plutôt qu'en termes de couches individuelles. Ainsi, un **BNN** N est constitué d'un certain nombre (disons, $m = q-1$) de blocs internes, suivis d'un bloc de sortie unique, noté O. Chaque bloc est composé d'une série de transformations linéaires et non linéaires. Le $k$ème bloc interne BLK_k ($k \in [m]$) dans un **BNN** peut être modélisé en tant qu'application

$$BLK_k : \{-1, 1\}^{d_k} \to \{-1, 1\}^{d_{k+1}}$$

associant à un vecteur de d_k valeurs dans $\{-1, 1\}$ un vecteur de d_{k+1} à valeurs dans $\{-1, 1\}$. Les entrées de BLK_1 sont les entrées de N (donc, $d_1 = d$, le nombre d'éléments dans X_d), les sorties de BLK_k ($k \in [m-1]$) sont les entrées de BLK_{k+1}, la sortie de BLK_m est l'entrée de O, et la valeur de sortie de O est la sortie de N. Alors que les entrées et sorties de N sont des vecteurs binaires, les couches internes de chaque bloc interne peuvent produire des sorties intermédiaires réelles. Une construction courante d'un bloc interne BLK_k ($k \in [m]$) est composée de trois opérations principales :

1. transformation linéaire (LIN) :
$$\boldsymbol{y} = \boldsymbol{A}_k \boldsymbol{x}_k + \boldsymbol{b}_k,$$
où $\boldsymbol{A}_k \in \{-1, 1\}^{d_{k+1} \times d_k}$ et $\boldsymbol{b}_k \in \mathbb{R}^{d_{k+1}}$
2. batch de normalisation (BN) :
$$z_i = \alpha_{k_i}\left(\frac{y_i - \mu_{k_i}}{\nu_{k_i}}\right) + \gamma_{k_i},$$
où $\boldsymbol{y} = (y_1, y_2, \ldots, y_{d_{k+1}}), \alpha_{k_i}, \mu_{k_i}, \nu_{k_i}, \gamma_{k_i} \in \mathbb{R}$,
et $\nu_{k_i} > 0$

3. binarisation (BIN) :

$$\boldsymbol{x}_{k+1} = sgn(\boldsymbol{z}),$$

où $\boldsymbol{z} = (z_1, z_2, \ldots, z_{d_{k+1}}) \in \mathbb{R}^{d_{k+1}}$,
et pour tout $i \in [d_{k+1}]$, $sgn(z_i) = 1$ if $z_i \geq 0$
et $sgn(z_i) = -1$ if $z_i < 0$,
de sorte que $\boldsymbol{x}_{k+1} \in \{-1, 1\}^{d_{k+1}}$

Le bloc de sortie produit la décision de classification. Il est composé de deux couches :

1. une transformation linéaire (LIN) :

$$\boldsymbol{w} = \boldsymbol{A}_q \boldsymbol{x}_q + \boldsymbol{b}_q, \text{ où } \boldsymbol{A}_q \in \{-1, 1\}^{s \times d_q} \text{ et } \boldsymbol{b}_q \in \mathbb{R}^s$$

2. une couche argmax (ARGMAX), qui produit en sortie l'indice le plus élevé de l'entrée la plus grande dans $\boldsymbol{w}$ en tant qu'étiquette prédite.

$$o = \text{argmax}_{i=1}^{s}\{w_i\}$$

Puisque nous nous intéressons à la classification binaire, nous supposons que le nombre de valeurs de sortie de N est $s = 2$ et que pour tout $\boldsymbol{x} \in \{-1, 1\}^d$, N classe $\boldsymbol{x}$ comme une instance positive si et seulement si $w_2 > w_1$ (la valeur de o est 2 dans ce cas, et 1 sinon).

Exemple 31. *Nous présentons via un exemple de **BNN** avec $q = 2$ blocs. Nous considérons un seul bloc interne, avec 4 entrées et 5 sorties. Les paramètres de N sont définis comme suit :*

LIN :

$$\boldsymbol{A}_1 = \begin{pmatrix} 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & 1 \end{pmatrix},$$

$$\boldsymbol{b}_1 = (-3.5, -3.5, -3.5, -3.5, -3.5)$$

BN :

$$\alpha_{\mathbf{1}} = \nu_{\mathbf{1}} = (1, 1, 1, 1, 1), \gamma_{\mathbf{1}} = \mu_{\mathbf{1}} = (0, 0, 0, 0, 0)$$

Le bloc de sortie O est défini par :

LIN :

$$\boldsymbol{A}_q = \begin{pmatrix} -1 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}, \boldsymbol{b}_q = (-4.5, 5)$$

5.2 Complexité des requêtes XAI pour les classifieurs booléens

Dans cette section, nous examinons successivement les 9 requêtes XAI répertoriées dans le tableau 4 et les présentons de manière plus détaillée (en les rendant formelles). Les problèmes de calcul correspondant à ces requêtes sont de différents types (décision, énumération, comptage, etc.). Étant donné que notre objectif n'est pas d'identifier de manière précise leurs complexités lorsqu'elles sont computationnellement difficiles, mais principalement d'affirmer qu'elles sont computationnellement difficiles lorsque tel est le cas, nous associons à chaque requête XAI un problème de décision qui suffit à montrer qu'une requête XAI « difficile » est effectivement exigeante sur le plan computationnel dans le cas général. Ceci est fait en démontrant que ce problème de décision est **NP-difficile au sens large**, ce qui signifie précisément que la conception d'un algorithme déterministe en temps polynomial pour résoudre la requête

XAI en question suffirait à prouver que P = NP. On se limite aussi au cas où une seule étiquette c est considérée. Pour les problèmes d'énumération, une source de complexité provient du nombre de sorties à fournir, qui peut être exponentiel par rapport à la taille de l'entrée. Nonobstant cette source de complexité, nous prouvons l'intraitabilité des problèmes d'énumération en montrant que générer une *seule* sortie est déjà NP-difficile. Nous montrons d'abord que chacune des 9 requêtes XAI est NP-difficile lorsqu'aucune restriction n'est imposée au classifieur booléen $\mathbf{C}$ considéré, ce qui est le choix fait à cette étape.

5.2.1 Le cas des requêtes de vérification

CIN : Comptage des instances associées à une classe donnée. Compter le nombre d'instances associées à la classe donnée correspondant à $\mathbf{C}$ est une requête de vérification utile. Lorsque le nombre trouvé diffère considérablement de celui attendu, cela peut refléter un problème avec l'ensemble de données utilisé pour apprendre le classifieur. Formellement :

Définition 45 (CIN). *Le problème **CIN** peut être formulé comme suit :*
- *Entrée : une représentation booléenne $\mathbf{C}$ sur X_d, et une classe cible $c \in \mathbb{B} = \{0, 1\}$ (instances positives ou négatives).*
- *Sortie : le nombre d'instances $\boldsymbol{x} \in \mathbb{B}^d$ classées par $\mathbf{C}$ en tant qu'instances positives si $c = 1$, ou en tant d'instances négatives si $c = 0$.*

Nous avons montré que **CIN** est intraitable [ABB+21] :

Proposition 6. ***CIN** est NP-difficile.*

EIN : Énumération des instances associées à une classe donnée. **EIN** est le problème d'énumération qui correspond à **CIN** :

Définition 46 (EIN). ***EIN** peut être formulé comme suit :*
- *Entrée : une représentation booléenne $\mathbf{C}$ sur X_d, et une classe cible $c \in \mathbb{B}$ (instances positives ou négatives).*
- *Sortie : énumérer avec un délai polynomial l'ensemble des instances positives $\boldsymbol{x} \in \mathbb{B}^d$ selon $\mathbf{C}$ si $c = 1$, et l'ensemble des instances négatives $\boldsymbol{x} \in \mathbb{B}^d$ selon $\mathbf{C}$ si $c = 0$.*

Le nombre d'instances positives (ou négatives) $\boldsymbol{x} \in \mathbb{B}^d$ selon $\mathbf{C}$ peut être exponentiel par rapport à la taille de l'entrée, de sorte que le temps nécessaire pour les calculer tous est probablement exponentiel également. En fait, il est peu probable qu'un algorithme en temps polynomial existe pour générer seulement un élément de cet ensemble. Pour le formaliser, soit **EIN** $[1]$ la restriction de **EIN** [AKM20] où la sortie consiste en une seule instance $\boldsymbol{x} \in \mathbb{B}^d$ classée par $\mathbf{C}$ comme instance positive lorsque $c = 1$ et qu'une telle instance existe, une seule instance $\boldsymbol{x} \in \mathbb{B}^d$ classée par $\mathbf{C}$ comme instance négative lorsque $c = 0$ et qu'une telle instance existe, et $\varnothing$ sinon.

Nous avons montré que **EIN** $[1]$ est intraitable [ABB+21] :

Proposition 7. ***EIN** $[1]$ est NP-difficile.*

IMA : Identification des attributs obligatoires / interdits dans une classe donnée. Lorsque la fréquence d'une attribut x_k (ou d'une combinaison d'attributs) dans la classe d'instances positives (ou négatives) associée à $\mathbf{C}$ est égale à 1, la présence de l'attribut / combinaison d'attributs est *obligatoire* pour qu'une instance soit reconnue comme appartenant à la classe, tandis que lorsqu'elle est égale à 0,

elle est *interdite*. Identifier les attributs obligatoires et interdits pour les classes d'instances positives (ou négatives) (telles qu'elles sont vues par le classifieur) est utile (le classifieur devrait être tel qu'il n'y ait aucune divergence entre ce qui est obtenu et ce qui est attendu). Formellement :

Définition 47 (IMA). ***IMA*** *peut être formulé comme suit :*

- *Entrée :* *une représentation booléenne* $\mathbf{C}$ *sur* X_d*, un terme* t *sur* X_d*, et une classe cible* $c \in \mathbb{B}$ *(instances positives ou négatives).*
- *Sortie :* 1 *si* t *est obligatoire pour la classe d'instances positives (resp. négatives) lorsque* $c = 1$ *(resp.* $c = 0$*), et* 0 *sinon.*

Nous avons montré que **IMA** est intraitable [ABB+21] :

Proposition 8. ***IMA*** *est* NP*-difficile.*

Une définition similaire peut être formulée pour les attributs interdites et le même résultat de complexité obtenu (vérifier si un attribut est interdit est également NP-difficile).

IIR : Identification des attributs irrelevants dans une classe donnée. Un attribut x_i est *irrelevant* pour la classe d'instances positives (resp. négatives) associée à $\mathbf{C}$ si et seulement si, pour chaque instance positive (resp. négative) $\boldsymbol{x}$ selon $\mathbf{C}$, l'instance $\boldsymbol{x}'$ qui coïncide avec $\boldsymbol{x}$ sur tous les attributs sauf x_i est également classifiée positivement (resp. négativement) par $\mathbf{C}$. Déterminer si un attribut est irrelevant ou non pour la classe d'instances positives (resp. négatives) associée à $\mathbf{C}$ est une requête de vérification utile pour identifier un biais de décision : il y a un tel biais lorsque l'appartenance de toute instance $\boldsymbol{x}$ à la classe associée à $\mathbf{C}$ dépend de sa valeur pour l'attribut x_i, alors que cela ne devrait pas être le cas. Formellement :

Définition 48 (IIR). ***IIR*** *peut être formulé comme suit :*

- *Entrée :* *une représentation booléenne* $\mathbf{C}$ *sur* X_d*, un attribut* x_i*, et une classe cible* $c \in \mathbb{B}$ *(instances positives ou négatives).*
- *Sortie :* 1 *si* x_i *est irrelevant pour la classe d'instances positives (resp. négatives) associée à* $\mathbf{C}$ *lorsque* $c = 1$ *(resp.* $c = 0$*), et* 0 *sinon.*

Nous avons montré que **IIR** est intraitable [ABB+21] :

Proposition 9. ***IIR*** *est* NP*-difficile.*

IMO : Identification des attributs monotones (ou anti-monotones) dans une classe donnée. Dans de nombreuses applications, on estime que l'augmentation de la valeur d'un certain attribut ne modifie pas l'appartenance à la classe des instances positives (resp. négatives) associée au classifieur booléen. En sens inverse, on pourrait également s'attendre à ce que la diminution de la valeur d'un autre attribut ne modifie pas non plus l'appartenance à la classe. Il est important de pouvoir tester si le classifieur $\mathbf{C}$ généré respecte ou non de telles hypothèses. Pour formaliser cela, une notion de monotonie (ou d'anti-monotonie) d'un classifieur est nécessaire, et peut être formulée comme suit : un classifieur $\mathbf{C}$ est *monotone* (resp. *anti-monotone*) par rapport à un attribut d'entrée x_i pour la classe des instances positives (resp. négatives), si pour toute instance positive (resp. négative) $\boldsymbol{x}$ selon $\mathbf{C}$, nous avons $\mathbf{C}(\boldsymbol{x}[x_i \leftarrow 1]) = 1$ (resp. $\mathbf{C}(\boldsymbol{x}[x_i \leftarrow 0]) = 1$). [10] Formellement :

Définition 49 (IMO). ***IMO*** *peut être formulé comme suit :*

10. Si $\boldsymbol{x} = (x_1, \cdots, x_d)$, alors $\boldsymbol{x}[x_k \leftarrow v]$ est le même vecteur que $\boldsymbol{x}$, sauf en x_k qui a pour valeur v.

— *Entrée :* une représentation booléenne $\mathbf{C}$ sur X_d, un attribut $x_i \in X_d$, et une classe cible $c \in \mathbb{B}$ (instances positives ou négatives).
— *Sortie :* 1 si $\mathbf{C}$ est monotone (resp. anti-monotone) par rapport à x_i pour la classe d'instances positives (resp. négatives), 0 sinon.

Nous avons montré que **IMO** est intraitable [ABB+21] :

Proposition 10. ***IMO*** *est* NP*-difficile.*

MCP : Mesurer la proximité d'une classe par rapport à un prototype. Enfin, on peut également être intéressé par la détermination du degré de conformité d'un prototype donné $\boldsymbol{x}$ à la classe que $\mathbf{C}$ lui associe, ce qui peut être évalué en calculant la distance de Hamming entre chaque élément de la classe d'instances positives (resp. négatives) et le prototype $\boldsymbol{x}$ lorsqu'il est classé comme une instance positive (resp. négative) par $\mathbf{C}$, et en considérant la distance maximale. Par convention, la distance maximale est fixée à ∞ lorsque la classe est vide. Plus la valeur est grande, plus la distance par rapport au prototype est grande. Lorsqu'un prototype de classe existe, il est censé être un élément « central » de la classe (c'est-à-dire minimisant la distance maximale par rapport à tout autre élément de la classe). Ainsi, une grande valeur peut indiquer un problème avec le classifieur qui a été appris. Formellement :

Définition 50 (**MCP**). ***MCP*** *peut être formulé comme suit :*
— *Entrée :* une représentation booléenne $\mathbf{C}$ sur X_d et une instance $\boldsymbol{x} \in \mathbb{B}^d$.
— *Sortie :* la distance de Hamming maximale de $\boldsymbol{x}$ par rapport à la classe associée à $\mathbf{C}$ (resp. le complément de cette classe) lorsque $\mathbf{C}(\boldsymbol{x}) = 1$ (resp. $\mathbf{C}(\boldsymbol{x}) = 0$).

Nous avons montré que **MCP** est intraitable [ABB+21] :

Proposition 11. ***MCP*** *est* NP*-difficile.*

5.2.2 Le cas des requêtes d'explication

EMC : Énumération d'explications de cardinalité minimum Étant donné une entrée $\boldsymbol{x}$ telle que $\mathbf{C}(\boldsymbol{x}) = c$, une *explication de cardinalité minimum* [SCD18b] de $\boldsymbol{x}$ est une instance d'entrée $\boldsymbol{x}'$ telle que $\mathbf{C}(\boldsymbol{x}') = c$, $\boldsymbol{x}'$ est cohérente avec $\boldsymbol{x}$ pour les valeurs 1 dans le sens que pour tout $i \in \{1, \ldots, d\}$, si $x'_i = 1$ alors $x_i = 1$, et $\boldsymbol{x}'$ a un nombre minimal de coordonnées égales à 1. En d'autres termes, les attributs qui sont définis à 1 dans $\boldsymbol{x}'$ suffisent à expliquer pourquoi $\boldsymbol{x}$ a été classé par $\mathbf{C}$ en tant qu'instance positive (ou négative). Formellement :

Définition 51 (**EMC**). ***EMC*** *peut être formulé comme suit :*
— *Entrée :* une représentation booléenne $\mathbf{C}$ sur X_d et une instance $\boldsymbol{x} \in \mathbb{B}^d$.
— *Sortie :* énumérer avec un délai polynomial l'ensemble de toutes les explications de cardinalité minimum de $\boldsymbol{x}$ étant donnée $\mathbf{C}$.

Le nombre d'explications de cardinalité minimum de $\boldsymbol{x}$ étant donnée $\mathbf{C}$ peut être exponentiel par rapport à la taille de l'entrée, donc le temps nécessaire pour les calculer est également probablement exponentiel. En fait, il est peu probable qu'un algorithme en temps polynomial existe pour générer une seule explication de cardinalité minimum. Pour le formaliser, soit **EMC**$[1]$ la restriction de **EMC** où la sortie consiste en une seule explication de cardinalité minimum de $\boldsymbol{x}$ étant donnée $\mathbf{C}$.

Nous avons montré que **EMC**$[1]$ est intraitable :

Proposition 12. ***EMC*** *[AKM20] est* NP*-difficile.*

DPI : Calcul d'une explication par impliquant premier. Étant donné une entrée $\boldsymbol{x}$ telle que $\mathbf{C}(\boldsymbol{x}) = c$, une *explication par impliquant premier* de $\boldsymbol{x}$ [SCD18b] (également appelée raison suffisante pour $\boldsymbol{x}$ étant donnée $\mathbf{C}$ [DH20]) est un sous-ensemble t minimal pour l'inclusion ensembliste de caractéristiques de $\boldsymbol{x}$ et qui satisfait la propriété que pour chaque extension $\boldsymbol{x}'$ de t sur X_d, nous avons $\mathbf{C}(\boldsymbol{x}') = c$. Formellement :

Définition 52 (DPI). ***DPI*** *peut être formulé comme suit :*
- *Entrée : une représentation booléenne* $\mathbf{C}$ *sur* X_d *et une instance* $\boldsymbol{x} \in \mathbb{B}^d$.
- *Sortie : une explication par impliquant premier de* $\boldsymbol{x}$ *étant donnée* $\mathbf{C}$.

Nous avons montré que **DPI** est intraitable [ABB+21] :

Proposition 13. ***DPI*** *est* NP*-difficile.*

ECO : Énumération des explications contrefactuelles. Les explications contrefactuelles sont nécessaires lorsque l'utilisateur est surpris par le résultat $\boldsymbol{y}$ fourni par le classifieur $\mathbf{C}$ pour une instance donnée $\boldsymbol{x}$. Nous avons $\mathbf{C}(\boldsymbol{x}) = 1$ (resp. $= 0$) tandis que l'utilisateur s'attendait à ce que $\mathbf{C}(\boldsymbol{x}) = 0$ (resp. $= 1$). Une *explication contrefactuelle* de $\boldsymbol{x}$ étant donnée $\mathbf{C}$ est une instance $\boldsymbol{x}'$ qui est aussi proche que possible de $\boldsymbol{x}$ en termes de distance de Hamming et telle que $\mathbf{C}(\boldsymbol{x}') \neq \mathbf{C}(\boldsymbol{x})$. S'il n'existe pas de $\boldsymbol{x}'$ tel que $\mathbf{C}(\boldsymbol{x}') \neq \mathbf{C}(\boldsymbol{x})$, alors aucune explication contrefactuelle de $\boldsymbol{x}$ étant donnée $\mathbf{C}$ n'existe. Lorsque $\boldsymbol{x}'$ existe, l'ensemble des attributs qui diffèrent entre $\boldsymbol{x}$ et $\boldsymbol{x}'$ peut être considéré comme une explication sur la raison pour laquelle $\boldsymbol{x}$ n'a pas été classé comme attendu par $\mathbf{C}$. Formellement :

Définition 53 (ECO). ***ECO*** *peut être formulé comme suit :*
- *Entrée : une représentation booléenne* $\mathbf{C}$ *sur* X_d *et une instance* $\boldsymbol{x} \in \mathbb{B}^d$.
- *Sortie : énumérer avec un délai polynomial l'ensemble de toutes les explications contrefactuelles de* $\boldsymbol{x}$ *étant donnée* $\mathbf{C}$.

Le nombre d'explications contrefactuelles de $\boldsymbol{x}$ étant donnée $\mathbf{C}$ peut être exponentiel par rapport à la taille de l'entrée, donc le temps nécessaire pour les calculer est également exponentiel. Comme cela est le cas pour les explications de cardinalité minimale, il est peu probable qu'un algorithme en temps polynomial existe pour générer une seule explication contrefactuelle. Nous considérons **ECO** [1] qui est la restriction d'**ECO** où la sortie consiste en une seule explication contrefactuelle de $\boldsymbol{x}$ étant donnée $\mathbf{C}$ lorsque une telle explication existe, et $\varnothing$ sinon.

Nous avons montré qu'**ECO** [1] est intraitable :

Proposition 14. ***ECO****[AKM20] est* NP*-difficile.*

5.3 Sur l'intelligibilité des classifieurs booléens

Nous sommes désormais en mesure d'évaluer l'intelligibilité computationnelle de chaque famille de classifieurs, parmi les arbres de décision, les classifieurs DNF, les listes de décision, les forêts aléatoires, les arbres boostés, les perceptrons multi-couches booléens et les réseaux de neurones binaires utilisant des attributs booléens. Cette intelligibilité est évaluée en déterminant l'ensemble des requêtes d'XAI (parmi les 9 considérées dans la section précédente) offertes par chaque famille, c'est-à-dire celles pour lesquelles le problème de calcul correspondant est réalisable en temps polynomial.

5.3.1 Sur l'intelligibilité des requêtes XAI pour les arbres de décision

Débutons par les arbres de décision (DT). Les arbres de décision sont largement reconnus comme l'un des modèles les plus interprétables en apprentissage automatique. Pour chaque famille de requêtes XAI que nous avons définies précédemment, nous pouvons examiner comment les arbres de décision se comportent. Par exemple, pour la requête CIN (Compter les Instances Associées à une Classe), les arbres de décision offrent un algorithme efficace. De même, nous pouvons évaluer la capacité des arbres de décision à fournir des explications et des réponses aux questions pertinentes sur leurs décisions de classification.

Proposition 15. *Pour chaque requête d'énumération parmi **EMC, ECO, EIN**, il existe un algorithme d'énumération avec un délai polynomial lorsque le classifieur booléen en question est un arbre de décision sur X_d. De plus, chaque requête parmi **DPI, CIN, IMA, IIR, IMO, MCP** est dans* P *lorsque le classifieur booléen en question est un arbre de décision sur X_d.*

Preuve 1. *D'après la définition, pour chacune des 9 requêtes XAI considérées, la classe cible peut être soit celle des instances positives, soit celle des instances négatives. Cela ne pose aucun problème pour les arbres de décision. En effet, pour tout arbre de décision T sur X_d, on peut calculer en temps linéaire un arbre de décision T' représentant la classe complémentaire de celle associée à T, c'est-à-dire un arbre de décision T' tel que $\forall \boldsymbol{x} \in \mathbb{B}^d$, $T'(\boldsymbol{x}) = 1$ si et seulement si $T(\boldsymbol{x}) = 0$. Pour obtenir T' à partir de T, il suffit de remplacer dans T chaque nœud feuille étiqueté par 1 par un nœud feuille étiqueté par 0, et chaque nœud feuille étiqueté par 0 par un nœud feuille étiqueté par 1.* [11] *Maintenant, [AKM20] ont identifié des conditions suffisantes pour qu'un classifieur (à étiquettes multiples, mais toujours booléen) offre des requêtes XAI traitables basées sur les requêtes et transformations du langage $\mathcal{L}$ utilisé pour le représenter. Ces requêtes et transformations sont des requêtes et transformations standards de la carte de compilation des connaissances [DM02].*

Il s'avère que le langage $\mathcal{DT}$ des arbres de décision sur des variables booléennes satisfait bon nombre de ces requêtes et transformations, à savoir **CO**, **CD**, **ME**, **CT**, **IM**, **OPT**, **EQ**, **SE**. *Cela a été démontré dans [KLMT13] pour chacune d'entre elles, à l'exception de* **OPT**. *Avant d'utiliser les résultats rapportés dans [AKM20], nous rappelons d'abord brièvement les définitions de ces requêtes et transformations.*

- ***Requêtes** [AKM20]*
  - ***Consistance** (**CO**) : un langage de représentation propositionnelle $\mathcal{L}$ satisfait* **CO** *si et seulement s'il existe un algorithme en temps polynomial qui associe chaque représentation Σ de $\mathcal{L}$ en entrée à 1 si Σ est consistant (i.e. satisfiable), et à 0 sinon.*
  - ***Implication** (**IM**) : un langage de représentation propositionnelle $\mathcal{L}$ satisfait* **IM** *si et seulement s'il existe un algorithme en temps polynomial qui associe chaque représentation Σ de $\mathcal{L}$ et chaque terme en entrée γ à 1 si $\gamma \models \Sigma$ est vrai, et à 0 sinon.*
  - ***Implication sémantique** (**SE**) : un langage de représentation propositionnelle $\mathcal{L}$ satisfait* **SE** *si et seulement s'il existe un algorithme en temps polynomial qui associe les représentations Σ_1 et Σ_2 de $\mathcal{L}$ en entrée à 1 si $\Sigma_1 \models \Sigma_2$ est vrai, et à 0 sinon.*
  - ***Équivalence** (**EQ**) : un langage de représentation propositionnelle $\mathcal{L}$ satisfait* **EQ** *si et seulement s'il existe un algorithme en temps polynomial qui associe les représentations Σ_1 et Σ_2 de $\mathcal{L}$ en entrée à 1 si $\Sigma_1 \equiv \Sigma_2$ est vrai, et à 0 sinon.*

11. Autrement dit, les arbres de décision satisfont la transformation ¬**C** de la carte de compilation des connaissances [KLMT13].

- ***Comptage de modèles* (CT)** *: un langage de représentation propositionnelle $\mathcal{L}$ satisfait* **CT** *si et seulement s'il existe un algorithme en temps polynomial qui associe chaque représentation Σ de $\mathcal{L}$ en entrée à un entier non négatif (en notation binaire) correspondant au nombre de modèles $||\Sigma||$ de Σ sur $Var(\Sigma)$.*
- ***Énumération de modèles* (ME)** *: un langage de représentation propositionnelle $\mathcal{L}$ satisfait* **ME** *si et seulement s'il existe un algorithme d'énumération avec un délai polynomial pour l'ensemble $[\Sigma]$ de modèles de Σ sur $Var(\Sigma)$. Ici, l'algorithme doit générer tous les modèles en séquence, sans qu'aucun modèle ne soit listé plus d'une fois. De plus, l'algorithme doit garantir que chaque délai entre la génération de deux modèles successifs et entre la génération du dernier modèle et la notification de la fin est polynomial en la taille de Σ.*

- ***Transformations [AKM20]***
  - ***Conditionnement* (CD)** *: un langage de représentation propositionnelle $\mathcal{L}$ satisfait* **CD** *si et seulement s'il existe un algorithme en temps polynomial qui associe chaque représentation Σ de $\mathcal{L}$ et chaque terme cohérent γ en entrée à une représentation de $\mathcal{L}$ qui est logiquement équivalente à $\Sigma|\gamma$.*

Définissons maintenant **OPT** *et expliquons pourquoi $\mathcal{DT}$ vérifie* **OPT**.

***Optimisation* (OPT).** *Un langage de représentation propositionnelle $\mathcal{L}$ satisfait* **OPT** *si et seulement s'il existe un algorithme en temps polynomial qui prend en entrée $\varphi \in \mathcal{L}$ et une application $w : Var(\varphi) \to \mathbb{Q}$ et qui retourne une représentation $\min_w(\varphi) \in \mathcal{L}$ dont les modèles sont les modèles ω de φ qui minimisent $w(\varphi) = \sum_{x \in Var(\varphi)} w(x).\omega(x)$.*

Montrons maintenant pourquoi $\mathcal{DT}$ vérifie **OPT**. *Soit $\varphi \in \mathcal{DT}$. Chaque sous-formule de φ correspond à un noeud unique de φ, la racine de la sous-formule en question.*
Définissons $w(N)$ et $\min_w(N)$ pour les noeuds N de φ par induction structurelle :

- ***Cas de base :** les feuilles de φ*
  - *si N est une feuille étiquetée à 0, alors $w(N) = +\infty$ car la sous-formule correspondante est contradictoire et $\min_w(N) = 0$.*
  - *si N est une feuille étiquetée à 1, alors soit V_N l'ensemble des variables de $Var(\varphi)$ privé des variables figurant sur le chemin de la racine de φ à N, on a $w(N) = \sum_{x \in V_N : w(x) < 0} w(x)$ et $\min_w(N)$ est un élément de $\mathcal{DT}$ équivalent au terme $\bigwedge_{x \in V_N : w(x) < 0} x \wedge \bigwedge_{x \in V_N : w(x) > 0} \overline{x}$.*
- ***Étape inductive :** les noeuds internes de φ*
  - *N est un noeud de la forme $N = ite(x, N_1, N_2)$ où $x \in Var(\varphi)$, on a $w(N) = \min(w(N_1), w(x) + w(N_2))$ et $\min_w(N)$*
    $= ite(x, \min_w(N_1), \min_w(N_2))$ *si* $w(N_1) = w(x) + w(N_2)$

    $= ite(x, \min_w(N_1), 0)$ *si* $w(N_1) < w(x) + w(N_2)$

    $= ite(x, 0, \min_w(N_2))$ *si* $w(N_1) > w(x) + w(N_2)$

$\min_w(\varphi)$ est obtenu en calculant $\min_w(rac_\varphi)$ où rac_φ est la racine de φ (on commence par calculer $w(N)$ pour chaque N des feuilles vers la racine, puis on calcule $\min_w(N)$ de la racine vers les feuilles). L'ensemble du calcul se fait en temps polynomial dans la taille de l'entrée.

Exemple 32. *Dans l'optique de rendre la preuve de* **OPT** *plus claire, nous introduisons un exemple illustratif. Considérons l'arbre φ et l'application w comme illustrés dans les figures 22 et 23.*

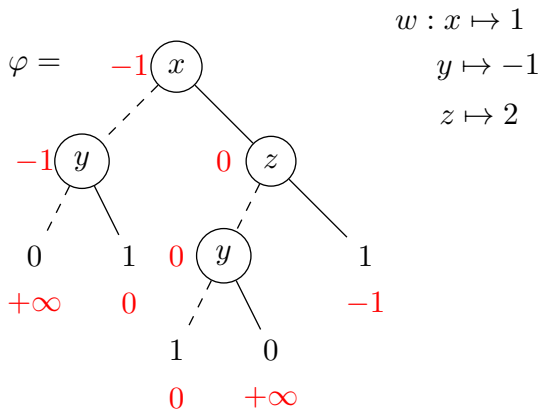


FIGURE 22 – L'arbre de décision φ. En rouge, les valeurs de $w(N)$ de chaque noeud.

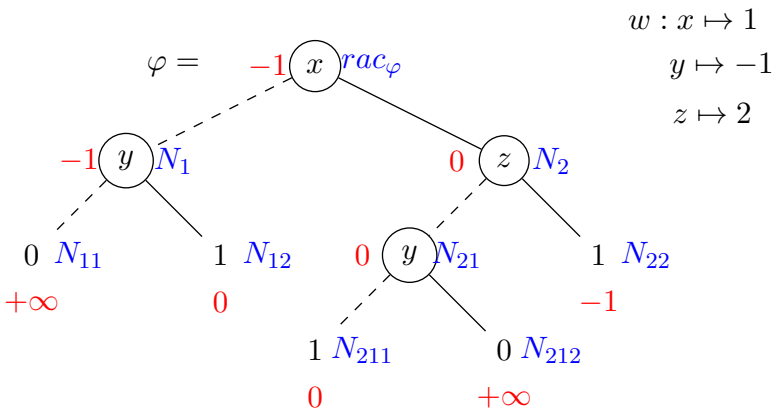
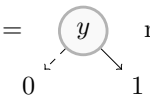
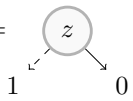
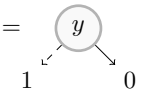
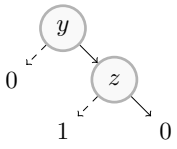
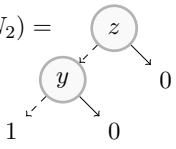


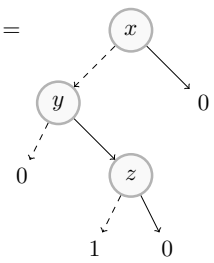
FIGURE 23 – En bleu : les identifiants de chaque noeud.

Les valeurs prises par $\min_w$ *en chaque noeud sont données ci-après.*

$\min_w(N_{11}) = 0$ $\min_w(N_{22}) =$  $\min_w(N_{12}) =$  $\min_w(N_{211}) = 1$ $\min_w(N_{212}) = 0$

$\min_w(N_{21}) =$  $\min_w(N_1) =$  $\min_w(N_2) =$ 

Finalement, nous avons

$\min_w(rac_\varphi) =$ 

Pour cet exemple, $\min_w(\varphi)$ *a un seul modèle* m*.* m *est tel que* $m[x] = 0, m[y] = 1$ *et* $m[z] = 0$*. Nous avons* $w(\varphi) = -1$*.*

*En utilisant les résultats rapportés dans [AKM20], nous concluons que les arbres de décision (*DT*) offrent les requêtes XAI* ***EMC, ECO, CIN, EIN, IMA et MCP****. Enfin, bien que les arbres de décision (*DT*) ne satisfont pas* **FO** *[KLMT13], les requêtes XAI qui ont été traitées en utilisant l'oubli dans [AKM20] nécessitent d'appliquer la transformation d'oubli* **FO***, pour éliminer les variables représentant les classes de* T*. Cela est inutile ici car aucune variable de classe n'est utilisée dans* T *(seules deux classes sont implicitement considérées ici, celle associée à* T*, alias la classe des instances positives, et son ensemble complémentaire qui peut être obtenu en calculant* T'*). Par conséquent, les arbres de décision (*DT*) offrent également les requêtes XAI* [12] ***DPI, IIR et IMO****.*

5.3.2 Sur l'intelligibilité des requêtes XAI pour les classifieurs booléens

Maintenant, tournons notre attention vers les autres classifieurs booléens. Contrairement aux arbres de décision, qui sont réputés pour leur interprétabilité, d'autres familles de modèles tels que les classifieurs DNF, les listes de décision, les forêts aléatoires, les arbres boostés, les perceptrons multi-couches booléens et les réseaux de neurones binaires présentent des défis plus complexes en matière d'intelligibilité. Nous allons explorer comment chacune de ces familles de modèles répond aux requêtes d'explicabilité que nous avons établies précédemment.

Proposition 16. *Chaque problème parmi* ***EMC****[1],* ***DPI, ECO****[1],* ***CIN, EIN****[1],* ***IMA, IIR, IMO, MCP*** *est* NP*-difficile lorsque le classifieur booléen en question est une formule* DNF*, une liste de décision, une forêt aléatoire, un arbre boosté, un perceptron multi-couches booléens ou un réseau de neurones binaire sur* X_d*.*

Preuve 2. *La preuve est organisée en trois parties. Dans une première partie, nous montrons que le problème SAT pour les formules* CNF *peut être réduit en temps polynomial à chaque problème parmi* ***EMC, DPI, ECO, CIN, EIN, IMA, IIR, IMO, MCP*** *où le classifieur booléen considéré* $\mathbf{C}$ *est donné sous forme d'une formule* CNF*. Dans une deuxième partie, nous montrons comment un classifieur* CNF *peut être associé en temps polynomial à un classifieur équivalent ayant la forme d'une liste de décision, d'une forêt aléatoire, d'un arbre boosté, d'un perceptron multi-couche booléen ou d'un réseau de neurones binaire sur des attributs booléens. En combinant les réductions polynomiales de la première partie avec les traductions polynomiales de la deuxième partie, les résultats de* NP*-difficulté énoncés dans la proposition et concernant les listes de décision, les forêts aléatoires, les arbres boostées, les perceptrons multi-couches booléens et les réseaux de neurones binaires suivent. Enfin, le cas des classifieurs* DNF *est abordé dans une troisième partie.*

- **EMC** $[1]$**.** *Soit* $\alpha = \bigwedge_{i=1}^{k} \delta_i$ *une formule* CNF *sur* $\{x_1, \ldots, x_{d-1}\}$*. Nous associons à* α *en temps polynomial le couple* $(\mathbf{C}, \boldsymbol{x})$ *où* $\mathbf{C} = \bigwedge_{i=1}^{k} \bigwedge_{j=1}^{d} (\delta_i \vee x_j)$ *est une formule* CNF *sur* $X_d = \{x_1, \ldots, x_d\}$ *(équivalente à* $\alpha \vee (\bigwedge_{i=1}^{d} x_i)$*), et* $\boldsymbol{x} = \bigwedge_{i=1}^{d} x_i$*. Clairement,* $\mathbf{C}$ *classe* $\boldsymbol{x}$ *comme une instance positive. Maintenant, il y a deux cas :*
  - *Si* α *est insatisfiable, alors* $\mathbf{C}$ *est équivalent à* $\bigwedge_{i=1}^{d} x_i$*. Dans ce cas, l'unique explication de cardinalité minimale pour* $\boldsymbol{x}$ *étant donnée* $\mathbf{C}$ *est égale à* $\boldsymbol{x}$*.*
  - *Si* α *est satisfiable, alors elle a un modèle* $\boldsymbol{x}'$ *sur* $\{x_1, \ldots, x_{d-1}\}$*. L'instance* $\boldsymbol{x}'' \in \mathbb{B}^d$ *qui étend* $\boldsymbol{x}'$ *et fixe* x_d *à* 0 *est classée comme une instance positive par* $\mathbf{C}$*, et elle contient moins*

12. Une preuve plus directe peut être trouvée dans [IIMS20].

d'attributs définis à 1 *que* $\boldsymbol{x}$*, donc* $\boldsymbol{x}$ *n'est pas une explication de cardinalité minimale pour* $\boldsymbol{x}$ *dans ce cas.*

— **DPI.** *Soit* α *une formule* CNF *sur* $\{x_1, \ldots, x_{d-1}\}$*. Nous associons à* α *en temps polynomial le couple* $(\mathbf{C}, \boldsymbol{x})$ *où* $\mathbf{C} = \alpha \wedge (\bigvee_{i=1}^{d} \overline{x_i})$ *est une formule* CNF *sur* $X_d = \{x_1, \ldots, x_d\}$ *et* $\boldsymbol{x} = \bigwedge_{i=1}^{d} x_i$*. Par construction,* $\boldsymbol{x}$ *est classifié par* $\mathbf{C}$ *comme une instance négative. Maintenant :*

- *Si* α *est insatisfiable, alors chaque instance* $\boldsymbol{x}' \in \mathbb{B}^d$ *est classifiée par* $\mathbf{C}$ *comme une instance négative puisque* $\mathbf{C}$ *est équivalent à* $\bot$*. Cela revient à dire qu'il existe une unique explication d'impliquant premier de* $\boldsymbol{x}$ *classifiée par* $\mathbf{C}$ *comme une instance négative, à savoir* $\top$*.*
- *Si* α *est satisfiable, alors elle a un modèle sur* $\{x_1, \ldots, x_{d-1}\}$*, et l'instance* $\boldsymbol{x}'$ *qui étend ce modèle et fixe* x_d *à* 0 *est un modèle de* $\alpha \wedge (\bigvee_{i=1}^{d} \overline{x_i})$*. Ainsi,* $\boldsymbol{x}'$ *est classifiée par* $\mathbf{C}$ *comme une instance positive, et par conséquent* $\top$ *n'est pas une explication d'impliquant premier de* $\boldsymbol{x}$ *étant donnée* $\mathbf{C}$*.*

— **ECO** [1]**.** *Soit* α *une formule* CNF *sur* $\{x_1, \ldots, x_{d-1}\}$*. Considérons la même réduction polynomiale que dans le cas* ***DPI****. Il y a deux cas :*

- *Si* α *est insatisfiable, alors chaque instance* $\boldsymbol{x}' \in \mathbb{B}^d$ *est classifiée par* $\mathbf{C}$ *comme une instance négative puisque* $\mathbf{C}$ *est équivalent à* $\bot$*. Ainsi, dans ce cas, il n'y a pas d'explication contrefactuelle de* $\boldsymbol{x}$ *donnée* $\mathbf{C}$*.*
- *Si* α *est satisfiable, alors elle a un modèle sur* $\{x_1, \ldots, x_{d-1}\}$*. L'instance* $\boldsymbol{x}' \in \mathbb{B}^d$ *qui étend ce modèle et fixe* x_d *à* 0 *est un modèle de* $\alpha \wedge (\bigvee_{i=1}^{d} \overline{x_i})$*. Dans ce cas, l'ensemble des instances positives données* $\mathbf{C}$ *n'est pas vide, et par conséquent, une explication contrefactuelle de* $\boldsymbol{x}$ *donnée* $\mathbf{C}$ *existe.*

— **CIN.** *Soit* α *une formule CNF. Nous associons à* α *en temps polynomial le couple* (ρ, c) *où* $\rho = \alpha$*, et* $c = 1$*. Le nombre d'instances* $x \in \mathbb{B}^d$ *classées positivement par* ρ *est égal au nombre de modèles de* α*. S'il était possible de calculer ce nombre en temps polynomial, alors il serait possible de décider en temps polynomial si* α *est satisfiable ou non.*

— **EIN [1].** *Nous considérons la même réduction polynomiale que dans le cas de* **CIN.** *Une instance* $x \in \mathbb{B}^d$ *classée par* ρ *en tant qu'instance positive existe et pourra être produite si et seulement si* α *est satisfiable.*

— **IMA.** *Soit* α *une formule* CNF *sur* $\{x_1, \ldots, x_{d-1}\}$*. Nous associons à* α *en temps polynomial le triplet* $(\mathbf{C}, t, c)$ *où* $\mathbf{C}$ *est la même formule que dans la preuve du cas* ***EMC*** [1]*,* $t = x_d$ *et* $c = 1$ *:*

- *Si* α *est insatisfiable, alors* $\boldsymbol{x} = \bigwedge_{i=1}^{n} x_i$ *est l'unique instance de* $\mathbb{B}^d$ *classée positivement par* $\mathbf{C}$*. Ainsi, chaque attribut de* $\boldsymbol{x}$ *(en particulier, ceux de* $t = x_d$*) est obligatoire pour la classe des instances positives.*
- *Si* α *est satisfiable, alors* α *a un modèle sur* $\{x_1, \ldots, x_{d-1}\}$ *et l'instance* $\boldsymbol{x}'$ *qui étend ce modèle et fixe* x_d *à* 0 *est classée positivement par* $\mathbf{C}$*, montrant que* $t = x_d$ *n'est pas obligatoire pour la classe des instances positives.*

Le cas des attributs interdits est similaire (il suffit de considérer $\mathbf{C} = \alpha \vee (\bigwedge_{i=1}^{d} \overline{x_i})$ *au lieu de* $\mathbf{C} = \alpha \vee (\bigwedge_{i=1}^{d} x_i)$ *:* t_{x_d} *est interdit pour la classe des instances positives si et seulement si* α *est insatisfiable).*

— **IIR.** *Soit* α *une formule* CNF *sur* $\{x_1, \ldots, x_{d-1}\}$*. Associons à* α *en temps polynomial le triplet* $(\mathbf{C}, x_d, c)$ *où* $\mathbf{C} = \alpha \wedge x_d$ *est une formule* CNF *sur* $X_d = \{x_1, \ldots, x_d\}$*, et* $c = 0$ *:*

- *Si α est insatisfiable, alors $\mathbf{C}$ l'est aussi, et x_d est un attribut sans importance pour la classe des instances négatives associée à $\mathbf{C}$.*
- *Si α est satisfiable, alors elle possède un modèle sur $\{x_1, \ldots, x_{d-1}\}$. Considérons l'instance $\boldsymbol{x}$ qui étend ce modèle et attribue à x_d la valeur 1. Alors $\mathbf{C}(\boldsymbol{x}) = 1$. Cependant, l'instance $\boldsymbol{x}' = \boldsymbol{x}[x_d \leftarrow 0]$, qui coïncide avec $\boldsymbol{x}$ sur chaque attribut sauf x_d, a vérifie $\mathbf{C}(\boldsymbol{x}') = 0$. Cela montre que x_d est pertinent pour la classe des instances négatives associée à $\mathbf{C}$.*

— **IMO.** *Soit α une formule* CNF *sur $\{x_1, \ldots, x_{d-1}\}$. Associons à α en temps polynomial le triplet $(\mathbf{C}, x_d, c)$ où $\mathbf{C} = \alpha \wedge \overline{x_d}$ est une formule* CNF *sur $X_d = \{x_1, \ldots, x_d\}$, et $c = 1$:*

- *Si α est insatisfiable, alors $\mathbf{C}$ l'est aussi et, en tant que tel, $\mathbf{C}$ est évidemment monotone par rapport à x_d (elle est monotone par rapport à chaque attribut).*
- *Si α est satisfaisable, alors elle possède un modèle sur $\{x_1, \ldots, x_{d-1}\}$. Considérons l'instance $\boldsymbol{x}$ qui étend ce modèle et attribue à x_d la valeur 0. Alors $\mathbf{C}(\boldsymbol{x}) = 1$. Cependant, l'instance $\boldsymbol{x}' = \boldsymbol{x}[x_d \leftarrow 1]$, qui coïncide avec $\boldsymbol{x}$ sur chaque attribut sauf x_d, a $\mathbf{C}(\boldsymbol{x}') = 0$. Cela montre que $\mathbf{C}$ n'est pas monotone par rapport à x_d pour la classe des instances positives.*

Le cas des attributs anti-monotones est similaire (il suffit de considérer $\mathbf{C} = \alpha \wedge x_d$ et d'étendre le contre-modèle de α en attribuant la valeur 1 à x_d lorsque α est satisfiable).

— **MCP.** *Soit α une formule* CNF *sur $\{x_1, \ldots, x_{d-1}\}$. Considérons la même réduction polynomiale que dans le cas **EMC** [1] :*

- *Si α est insatisfiable, alors l'unique instance classée positivement par $\mathbf{C}$ est $\boldsymbol{x}$, ce qui montre que la distance de Hamming maximale de $\boldsymbol{x}$ à un élément de la classe associée à $\mathbf{C}$ est 0.*
- *Si α est satisfiable, alors elle possède un modèle sur $\{x_1, \ldots, x_{d-1}\}$ et l'instance $\boldsymbol{x}'$ qui étend ce modèle et fixe x_d à 0 est classée positivement par $\mathbf{C}$. Dans ce cas, la distance de Hamming maximale de $\boldsymbol{x}$ à la classe associée à $\mathbf{C}$ est ≥ 1.*

Maintenant, nous présentons la deuxième partie de la preuve :

— **Listes de décision.** *Chaque formule* CNF *peut être transformée en temps linéaire en une liste de décision équivalente $\mathbf{L}$ (voir le théorème 1 de [Riv87]). En conséquence, chaque réduction mentionnée dans la première partie de la preuve peut être transformée en une réduction de telle sorte que la représentation visée soit une liste de décision, ce qui conclut la preuve.*

— **Forêts aléatoires.** *Nous exploitons la même idée que dans la preuve du cas des listes de décision. Pour obtenir le résultat, il suffit de montrer que chaque formule* CNF *peut être transformée en temps linéaire en une forêt aléatoire équivalente F. La traduction se fait comme suit : étant donné une formule* CNF*, $\mathbf{C} = \bigwedge_{i=1}^{m} \delta_i$ sur X_d, nous lui associons en temps linéaire la forêt aléatoire*

$$F = \{T_1, \ldots, T_m, \underbrace{0, \ldots, 0}_{m-1}\}$$

sur X_d, où chaque T_i ($i \in [m]$) est un arbre de décision sur X_d qui représente la clause δ_i.

Chaque T_i ($i \in [m]$) est un arbre en forme de peigne qui peut être généré facilement en temps linéaire par rapport à la taille de δ_i : si δ_i est la clause vide, alors retournez $T_i = 0$; sinon, en considérant les littéraux ℓ de δ_i en séquence, générez un nœud de décision de la forme $ite(x, 1, T_i^\ell)$ (resp. $ite(x, T_i^\ell, 1)$ si ℓ est un littéral négatif $\overline{x}$ (resp. un littéral positif x)), où T_i^ℓ est un arbre de décision pour la clause $\delta_i \setminus \{\ell\}$.

Enfin, par construction, le seul sous-ensemble d'arbres de F qui contient plus de la moitié des arbres et qui peut être cohérent est $\{T_1, \ldots, T_m\}$ (tout autre sous-ensemble de F contenant au

moins m arbres contient un arbre réduit à 0, et en tant que tel, il est incohérent). En conséquence, F est équivalent à la conjonction de tous les arbres de $\{T_1, \ldots, T_m\}$. Comme chaque T_i ($i \in [m]$) est équivalent à la clause δ_i de $\mathbf{C}$, nous obtenons que F est équivalent à $\mathbf{C}$, comme attendu.

— **Arbres boostés.** *Directement depuis la preuve du cas des forêts aléatoires, étant donné qu'une forêt aléatoire $F = \{T_1, \ldots, T_m\}$ peut être transformée en temps linéaire en un arbre boosté équivalent $B = \{\langle T_1, \frac{1}{m}\rangle, \ldots, \langle T_m, \frac{1}{m}\rangle\}$ où le poids de chaque arbre est égal à $\frac{1}{m}$.*

— **Perceptrons multi-couches booléens.** *Il n'est pas difficile de transformer en temps polynomial toute formule $\texttt{CNF}$ $\mathbf{C} = \bigwedge_{i=1}^{k} \delta_i$ sur $X_d = \{x_1, \ldots, x_d\}$, telle que $\mathbf{C}$ ne contienne aucune clause valide (ceci peut être assuré efficacement), en un perceptron multi-couches booléen P sur X_d, qui est logiquement équivalent à $\mathbf{C}$. On utilise seulement trois couches : la première couche V_1 contient d sommets (un par variable $x_i \in X_d$), la dernière couche V_3 contient un seul sommet $v_{3,1}$, et la deuxième couche V_2 contient k sommets, un par clause de $\mathbf{C}$. La sortie de $v_{3,1}$ est la sortie de P. Soit δ_i une clause quelconque de $\mathbf{C}$ et soit $v_{2,i}$ le sommet correspondant. Pour chaque sommet $v_{1,j}$ ($j \in [d]$) de la première couche V_1 qui est associé à une variable $x_j \in X_d$, l'arc $(v_{1,j}, v_{2,i}) \in E$ qui relie $v_{1,j}$ à $v_{2,i}$ est étiqueté par $w(v_{1,j}, v_{2,i}) = 0$ si x_j n'apparaît pas dans δ_i, par $w(v_{1,j}, v_{2,i}) = 1$ si x_j est un littéral positif de δ_i, et par $w(v_{1,j}, v_{2,i}) = -1$ si $\neg x_j$ est un littéral négatif de δ_i. La valeur du biais $b_{2,i}$ est le nombre de littéraux négatifs dans δ, moins 1. Par construction, nous avons $o_{v_{2,i}}(\boldsymbol{x}) = 1$ si et seulement si $\boldsymbol{x}$ satisfait la clause δ_i associée à $v_{2,i}$. Maintenant, pour chaque sommet $v_{2,i}$ ($i \in [k]$) de la deuxième couche V_2, l'arc $(v_{2,i}, v_{3,1}) \in E$ qui relie $v_{2,i}$ à $v_{3,1}$ est étiqueté par $w(v_{2,i}, v_{3,1}) = 1$, et le biais $b_{3,1}$ est fixé à $-k$ En conséquence, la sortie de $o_{v_{3,1}}$ de P est 1 si et seulement si chaque clause δ_i de $\mathbf{C}$ est satisfaite par $\boldsymbol{x}$, ou en d'autres termes, si et seulement si $\boldsymbol{x}$ est un modèle de $\mathbf{C}$.*

— **Réseaux de neurones binaires.** *À une formule $\texttt{CNF}$, $\mathbf{C} = \bigwedge_{i=1}^{k} \delta_i$ sur $X_d = \{x_1, \ldots, x_d\}$, telle que $\mathbf{C}$ ne contienne aucune clause valide, nous associons en temps polynomial un réseau de neurones binaire (BNN) N avec $2d$ entrées dans $\{-1, 1\}$ et une valeur de sortie dans $\{1, 2\}$ telle que pour tout $\boldsymbol{x} \in \mathbb{B}^d$, $\boldsymbol{x}$ est un modèle de $\mathbf{C}$ si et seulement si sa traduction $transl(\boldsymbol{x}) \in \{-1, 1\}^{2d}$ est classée comme une instance positive par N (c'est-à-dire que la valeur de sortie de N est 2).*

La fonction de traduction $transl : \mathbb{B}^d \to \{-1, 1\}^{2d}$ peut être calculée en temps linéaire et est définie comme suit :

$$transl(x_1, \ldots, x_d) = (\underbrace{2x_1 - 1, 2x_1 - 1}_{2}, \ldots, \underbrace{2x_n - 1, 2x_n - 1}_{2})$$

c'est-à-dire, pour $i \in \{0, \ldots, d-1\}$, les $(2i+1)^e$ et $(2(i+1))^e$ coordonnées de $transl(x_1, \ldots, x_d)$ sont égales à $2x_{i+1} - 1$. N se compose d'un seul bloc intermédiaire, c'est-à-dire $m = 1$, de sorte que le nombre total de blocs est $q = 2$. Le nombre d'entrées du premier bloc est $2d$ et le nombre de sorties de ce bloc est k, le nombre de clauses de $\mathbf{C}$.

L'idée clé de notre traduction de $\texttt{CNF}$ en BNN est de considérer individuellement chacune des k clauses δ de $\mathbf{C}$ et de calculer la différence entre le nombre de littéraux de δ falsifiés par $\boldsymbol{x}$ et le nombre de littéraux de δ satisfaits par $\boldsymbol{x}$, qui est noté comme suit :

$$diff(\delta, \boldsymbol{x}) = |\{\ell \in \delta : \boldsymbol{x} \models \overline{\ell}\}| - |\{\ell \in \delta : \boldsymbol{x} \models \ell\}|.$$

On vérifie que $\boldsymbol{x}$ ne satisfait pas δ précisément lorsque $diff(\delta, \boldsymbol{x})$ est le nombre de littéraux dans δ.

*La première opération réalisée par le **BNN** est une transformation linéaire. Ainsi, on doit définir $\boldsymbol{A}_1$ et $\boldsymbol{b}_1$. Fondamentalement, on souhaite utiliser cette transformation pour stocker dans la sortie $\boldsymbol{y} = \boldsymbol{A}_1 transl(\boldsymbol{x}) + \boldsymbol{b}_1$ des informations sur la satisfaction des clauses de $\mathbf{C}$ par $\boldsymbol{x}$, de sorte que y_i ($i \in [k]$) corresponde à la clause δ_i de $\mathbf{C}$. Comme les clauses ne sont généralement pas construites sur toutes les variables, nous devons ajouter un mécanisme pour éviter de considérer les variables qui n'apparaissent pas dans une clause. Cela est réalisé dans $transl$ en dupliquant les coordonnées du vecteur d'entrée $\boldsymbol{x}$ (en plus de les traduire de $\mathbb{B}$ dans $\{-1, 1\}$). En effet, si un littéral ℓ sur $x \in X_d$ n'est pas présent dans une clause δ_i, sa contribution à y_i doit être de 0, ce qui est obtenu en multipliant les deux coordonnées associées à x dans $transl(\boldsymbol{x})$ par -1 et 1, respectivement. Si un littéral ℓ sur $x \in X_d$ apparaît dans δ_i, nous voulons que sa contribution à y_i soit définie à 1 si le littéral falsifie la clause et à -1 s'il la satisfait. En additionnant la contribution de chaque littéral de cette manière, nous obtenons le résultat attendu.*

Formellement, la matrice $\boldsymbol{A}_1[i]$ pour $i \in [k]$ est définie comme :

$$\boldsymbol{A}_1[i] = (\tau^1_{x_1}, \tau^2_{x_1}, \tau^1_{x_2}, \tau^2_{x_2}, \ldots, \tau^1_{x_d}, \tau^2_{x_d}), \text{ où}$$

$$\text{pour chaque } j \in [d] \begin{cases} \tau^1_{x_j} = -\tau^2_{x_j} & \text{si } x_j \notin Var(\delta_i) \\ \tau^1_{x_j} = \tau^2_{x_j} = 1 & \text{si } \overline{x_j} \in \delta_i \\ \tau^1_{x_j} = \tau^2_{x_j} = -1 & \text{si } x_j \in \delta_i \end{cases}$$

Ensuite, on définit $\boldsymbol{b}_1[i]$ ($i \in [k]$) à $-2 \times |\delta_i| + 1$. Nous obtenons que pour chaque $i \in [k]$, $y_i = 1$ si la clause δ_i est falsifiée par $\boldsymbol{x}$ et $y_i < 0$ si δ_i est satisfaite par $\boldsymbol{x}$. Quant à la normalisation par lots (batch normalization), pour chaque $i \in [k]$, nous fixons les paramètres $\alpha_{1_i} = \nu_{1_i}$ à 1 et $\mu_{1_i} = \gamma_{1_i}$ à 0, de sorte que la sortie de la transformation coïncide avec son entrée. Enfin, la transformation de binarisation a lieu. Clairement, la sortie du bloc interne de N est un vecteur $\boldsymbol{x}' = (x'_1, \ldots x'_k) \in \{-1, 1\}^k$ tel que pour chaque $i \in [k]$, $x'_i = 1$ si $\boldsymbol{x}$ falsifie δ_i et $x'_i = -1$ si $\boldsymbol{x}$ satisfait δ_i.

Par construction, ce vecteur $\boldsymbol{x}'$ est l'entrée du bloc de sortie O de N. Rappelons que la valeur de sortie o de O est la sortie de N, c'est une valeur dans $\{1, 2\}$ indiquant si l'entrée $transl(\boldsymbol{x})$ est classée positivement par N. La transformation linéaire utilisée dans O est donnée par $\boldsymbol{A}_q \in \{-1, 1\}^{2 \times k}$ et $\boldsymbol{b}_q \in \mathbb{R}^2$ tels que $\boldsymbol{A}_q[1] = (\underbrace{1, \ldots, 1}_{k})$ et $\boldsymbol{A}_q[2] = \underbrace{(-1, \ldots, -1)}_{k}$, tandis que $\boldsymbol{b}_q[1] = 2k$ et $\boldsymbol{b}_q[2] = 1$. Ainsi, lorsque l'instance $\boldsymbol{x}$ satisfait $\mathbf{C}$, elle satisfait chaque clause et la coordonnée w_2 de la sortie $\boldsymbol{w}$ de la transformation linéaire $\boldsymbol{w} = \boldsymbol{A}_q \boldsymbol{x}' + \boldsymbol{b}_q$ est égale à $k + 1$, tandis que sa coordonnée w_1 est égale à k. En revanche, lorsque l'instance $\boldsymbol{x}$ ne satisfait pas $\mathbf{C}$, nous avons $w_2 \leq k$ et $w_1 \geq k + 1$. En conséquence, la dernière transformation de O (la couche ARGMAX) envoie $o = 2$ lorsque $\boldsymbol{x}$ satisfait $\mathbf{C}$ et $o = 1$ sinon. Ainsi, N peut être considéré comme une représentation de la formule `CNF` *$\mathbf{C}$ modulo la fonction de traduction $transl$, au sens où pour tout $\boldsymbol{x} \in \mathbb{B}^d$, $\mathbf{C}(\boldsymbol{x}) = c$ si et seulement si $N(transl(\boldsymbol{x})) = c + 1$.*

Finalement, à chaque réduction donnée dans la première partie de la preuve, nous pouvons associer une autre réduction polynomiale où $\mathbf{C}$ est un classifieur `DNF`*. Cela découle de deux points : (1) la dualité entre les classifieurs* `CNF` *et les classifieurs* `DNF` *qui établit que $\boldsymbol{x} \in \mathbb{B}^d$ est une instance positive d'un concept représenté par un classifieur* `CNF`*, $\mathbf{C}$ sur X_d si et seulement si $\boldsymbol{x}$ est une instance négative du concept (complémentaire) représenté par un classifieur* `DNF`*, D équivalent à $\neg\mathbf{C}$ (évidemment, un tel D est calculable à partir de $\mathbf{C}$ en temps linéaire en appliquant les lois de De Morgan) et (2) le fait que toutes les requêtes d'explicabilité que nous avons considérées doivent être capables de prendre en compte à la fois les instances positives et négatives. Ceci conclut la preuve.*

Les deux dernières propositions montrent ainsi l'existence d'un *grand écart d'intelligibilité computationnelle* entre les familles de classifieurs en question. Puisque chacune des 9 requêtes d'explicabilité XAI est traitable pour la famille des arbres de décision, *les arbres de décision peuvent être considérés comme très intelligibles en comparaison aux autres familles de classifieurs considérées dans ce chapitre*. À l'extrême opposée du spectre, les classifieurs DNF, les listes de décision, les forêts aléatoires, les arbres boostées, les perceptrons multi-couches booléens et les réseaux neurones binaires apparaissent comme peu intelligibles puisque aucune des 9 requêtes XAI n'est traitable pour l'une de ces familles.

Il est à noter que les résultats rapportés dans les deux dernières propositions diffèrent considérablement de ceux présentés dans un certain nombre d'articles précédents, où l'on cherche à trouver une traduction en temps polynomial préservant l'équivalence (alias un codage) d'une famille donnée $\mathcal{L}$ de classifieurs booléens vers des formules CNF (voir par exemple [Nar18, NKR+18]).

Déterminer de telles traductions permet de tirer parti des techniques de raisonnement automatisé pour répondre aux requêtes XAI, étant donné que les solveurs existants pour les représentations booléennes sont le plus souvent basés sur le format CNF. Ici, nous avons cherché des traductions en temps polynomial du langage des formules CNF vers les langages $\mathcal{L}$ des classifieurs sur lesquels nous nous concentrons, afin de prouver que les requêtes XAI sont computationnellement difficiles lorsque le classifieur d'entrée est dans $\mathcal{L}$. C'est une perspective différente, mais complémentaire. En effet, lorsque de telles traductions existent, exploiter les codages CNF pour répondre aux requêtes XAI a du sens d'un point de vue computationnel. Cependant, ces traductions ne sont pas garanties d'exister pour chaque famille $\mathcal{L}$ de classifieurs, de sorte qu'il peut arriver qu'une requête XAI soit traitable pour $\mathcal{L}$ tout en étant computationnellement difficile pour les classifieurs CNF. Notre étude montre que c'est précisément ce qui se passe avec les arbres de décision. Pour cette famille de classifieurs, il est pertinent de développer des algorithmes pour répondre aux requêtes XAI qui sont directement basés sur les représentations en question (arbres de décision), au lieu de concevoir des codages CNF.

5.4 Discussion

Notre étude a exploré diverses requêtes d'explication et de vérification, telles que résumées dans la table 4, et considéré plusieurs familles de classifieurs. Nous n'avons pas cherché à être exhaustifs dans le choix des familles de classifieurs ni dans le choix des requêtes. Naturellement, des résultats complémentaires peuvent être obtenus si l'on examine d'autres familles de classifieurs et d'autres requêtes. Par exemple, des résultats de polynomialité (appartenance à P) ont été observés pour certaines requêtes lorsque l'on considère les fonctions linéaires à seuil (voir [MSGC+20]). Et dans une moindre mesure les classifieurs monotones (voir [MGC+21]) et et les arbres de décision multivariées (voir [CCMS23]). Ces résultats complètent nos travaux présentés dans ce chapitre et ouvrent la voie à des recherches futures.

Dans ce chapitre, nous n'avons pas cherché à séparer les cas où la classe c (donnée en entrée) est positive (« + ») ou négative (« − »). Les auteurs de [CMS23] ont montré que pour certaines familles de classifieurs, expliquer une décision positive peut être en temps polynomial (dans P) mais expliquer une décision négative est NP-difficile. Il est bien sûr possible de construire de nouvelles requêtes qui fixent le choix de la classe et de chercher si, pour certaines familles de classifieurs, les problèmes relatifs à ces requêtes sont dans P ou NP-difficiles.

5.5 Conclusion

Dans ce chapitre, nous avons examiné sous un angle computationnel l'intelligibilité de plusieurs familles de classifieurs booléens : **les arbres de décision, les formules** `DNF`**, les listes de décision, les forêts aléatoires, les arbres boostés, les perceptrons multi-couches booléens et les réseaux de neurones binaires.** L'intelligibilité computationnelle d'une famille de classifieurs a été évaluée comme l'ensemble des requêtes XAI qui sont traitables lorsque le classifieur en question appartient à la famille. En considérant un ensemble de 9 requêtes XAI de base, nous avons montré l'existence d'un grand écart d'intelligibilité computationnelle entre les familles de classifieurs. En gros, les résultats obtenus montrent que, alors que les arbres de décision sont très intelligibles, les autres familles de classifieurs sur lesquelles nous nous sommes concentrés ne sont pas du tout intelligibles. Cela concorde avec l'intuition largement partagée selon laquelle « les arbres de décision sont interprétables et les autres classifieurs d'apprentissage automatique ne le sont pas », mais plus que cela, nos résultats donnent une base formelle à cette intuition. Ce travail complète l'étude [AKM20] qui se concentre sur la conception de cas traitables pour un sur-ensemble des 9 requêtes XAI considérées ici, en utilisant des techniques de compilation de connaissances. Le fait que chacune des 9 requêtes XAI soit intraitable (NP-difficile) lorsque le classifieur booléen $\mathbf{C}$ considéré n'est pas contraint justifie le besoin de rechercher des représentations spécifiques des circuits dans des langages garantissant la traitabilité de ces requêtes, et des techniques de traduction (« compilation de connaissances ») pour transformer les classifieurs en représentations de langages traitables, comme cela a été fait dans [AKM20].

Les résultats obtenus ont confirmé le haut niveau d'interprétabilité des arbres de décision. Cependant, les arbres de décision peuvent également présenter des problèmes d'explicabilité liés au nombre d'explications possibles mais aussi à la taille des explications. Nous nous concentrons sur ces limitations dans les prochains chapitres du mémoire.

6

Le pouvoir explicatif des arbres de décision

Sommaire

6.1 Explications abductives et contrastives 102
6.1.1 Transformation d'arbres de décision binaires en forme normale 102
6.1.2 Explications abductives 103
6.1.3 Explications contrastives 104
6.2 Calculer toutes les explications abductives 105
6.2.1 Énumérer toutes les raisons suffisantes minimales 107
6.2.2 Synthétiser l'ensemble des raisons suffisantes 111
6.3 Calculer toutes les explications contrastives 115
6.4 Expérimentations 118
6.5 Conclusion 124

Dans ce chapitre, nous examinons la capacité computationnelle des arbres de décision booléens dans un contexte d'explication. Nous nous concentrons à la fois sur les explications abductives et sur les explications contrastives. Plus précisément, nous nous intéressons à la dérivation, à la minimisation et au comptage des explications abductives et contrastives. Nous démontrons que l'ensemble de toutes les explications abductives non redondantes (c'est-à-dire explications d'impliquants premiers, alias les raisons suffisantes) pour une instance étant donnée un arbre de décision peut être exponentiellement plus grand que la taille de l'entrée (l'instance et l'arbre de décision). Par conséquent, générer l'ensemble complet des raisons suffisantes pour une instance peut être hors de portée. De plus, calculer une seule raison suffisante ne suffit pas en général. En effet, deux raisons suffisantes pour une même instance peuvent différer sur tous leurs attributs. Pour faire face à ce problème et générer des vues synthétiques de l'ensemble de toutes les raisons suffisantes, nous introduisons les notions d'attribut pertinent et d'attribut nécessaire qui caractérisent les attributs apparaissant dans au moins une ou dans chaque raison suffisante pour une instance, et nous montrons qu'ils peuvent être calculées en temps polynomial. Nous introduisons également la notion d'importance explicative, qui indique à quelle fréquence chaque attribut apparaît dans l'ensemble de toutes les raisons suffisantes. Nous montrons comment le pouvoir explicatif d'un attribut et le nombre de raisons suffisantes pour une instance peuvent être obtenus via un comptage de modèles, ce qui se relève faisable en pratique dans de nombreux cas. Nous expliquons également comment énumérer les raisons suffisantes de taille minimale (alias les raisons suffisantes minimales). Enfin, nous montrons que, contrairement aux raisons suffisantes, l'ensemble de toutes les explications contrastives pour une instance donnée et un arbre de décision peut être dérivé, minimisé et énuméré en temps polynomial.

Ces travaux ont donné lieu à deux publications [ABB+22d, ABB+22c].

Introduction. D'abord, on peut noter que chaque instance possède au moins une explication abductive étant donné un classifieur booléen et qu'elle a également au moins une explication contrastive à moins que chaque instance soit positive ou que chaque instance soit négative. En particulier, chaque modèle d'apprentissage automatique associe à toute instance donnée $\boldsymbol{x}$ une explication abductive, que nous appelons la *raison directe* pour $\boldsymbol{x}$, qui est induite par la manière dont le modèle classe $\boldsymbol{x}$. Cette raison directe peut coïncider avec $\boldsymbol{x}$ (dans ce cas, elle n'est pas très utile), mais elle peut également être plus simple que $\boldsymbol{x}$, comme c'est le cas pour les arbres de décision. Rappelons que, bien qu'il n'y ait pas de notion formelle d'*interprétabilité* [Lip18], pour les problèmes de classification, les *arbres de décision* [BFOS84, Qui86] sont sans doute parmi les modèles d'apprentissage automatique les plus interprétables. En raison de leur interprétabilité, les arbres de décision sont souvent considérés comme des modèles cibles pour distiller un modèle boîte noire en un modèle compréhensible. De plus, les arbres de décision sont souvent les composants de choix pour construire des classifieurs ensemblistes (moins interprétables, mais potentiellement plus précis), tels que les forêts aléatoires [Bre01] et les arbres boostés [SCD16].

L'interprétabilité des arbres de décision est dotée de deux caractéristiques clés. D'une part, les arbres de décision sont *transparents* : chaque nœud dans un arbre de décision a une signification, et les principes utilisés pour générer tous les nœuds peuvent être expliqués. D'autre part, les arbres de décision sont *explicables localement* : étant donné un arbre de décision T, toute instance d'entrée $\boldsymbol{x}$ est associée à un chemin unique de la racine à la feuille qui conduit à une étiquette de décision. Le sous-ensemble d'attributs (positifs et négatifs) $t_{\boldsymbol{x}}^T$ qui apparaissent dans le chemin utilisé pour trouver l'étiquette correcte 1 ou 0 pour $\boldsymbol{x}$ dans l'arbre de décision T est appelé l'explication restreinte au chemin pour $\boldsymbol{x}$ [IIMS20], et c'est la raison directe de la classification de $\boldsymbol{x}$ en tant qu'instance positive ou négative donnée T. $t_{\boldsymbol{x}}^T$ est une explication abductive pour $\boldsymbol{x}$ donnée T, qui explique pourquoi $\boldsymbol{x}$ a été classifié par T comme il l'a été. En effet, chaque instance $\boldsymbol{x}'$ qui coïncide avec $\boldsymbol{x}$ sur $t_{\boldsymbol{x}}^T$ est classée par T de la même manière que $\boldsymbol{x}$. Cependant, de telles raisons directes peuvent contenir un nombre arbitraire d'attributs redondants [IIMS20]. Ceci motive la prise en compte d'autres types d'explications abductives dans le cas des arbres de décision, à savoir les raisons suffisantes [DH20] et les raisons suffisantes de taille minimale.

6.1 Explications abductives et contrastives

Nous nous concentrons tout d'abord sur la manière de dériver des explications à partir d'arbres de décision. Notre étude inclut les explications abductives et les explications contrastives. En résumé, il s'agit d'explorer comment les arbres de décision expliquent leurs décisions en mettant en avant les facteurs décisifs qui influent sur ces choix.

6.1.1 Transformation d'arbres de décision binaires en forme normale

Nous rappelons qu'un *arbre de décision binaire* (booléen) sur X_d est un arbre binaire dans lequel chaque nœud interne est étiqueté par l'une des d variables booléennes d'entrée, et chaque feuille est étiquetée soit par 0 soit par 1. Chaque variable apparaît au maximum une fois sur n'importe quel chemin de la racine à une feuille. La valeur $T(\boldsymbol{x}) \in \{0, 1\}$ de T pour l'instance d'entrée $\boldsymbol{x}$ est déterminée par l'étiquette de la feuille atteinte en suivant le chemin depuis la racine : à chaque nœud, on suit le fils gauche ou droit en fonction de la valeur d'entrée correspondante de la variable, soit 0 soit 1. La taille de T, notée $|T|$, correspond au nombre de nœuds qu'il contient. La classe des arbres de décision sur X_d est notée $\mathtt{DT}_d$.

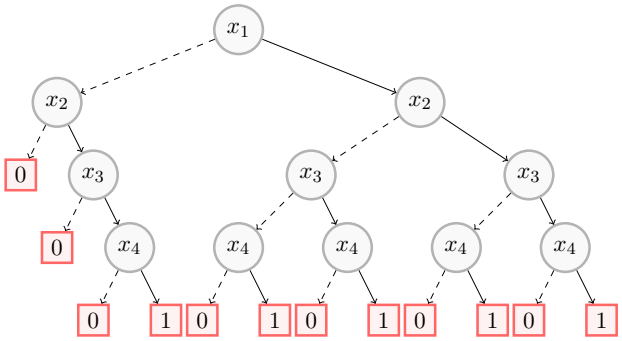


FIGURE 24 – Un arbre de décision T pour reconnaître les orchidées Cattleya.

Il est bien connu que tout arbre de décision $T \in \mathtt{DT}_d$ peut être transformé en temps linéaire en une disjonction de termes équivalente, notée $\mathtt{DNF}(T)$, où chaque terme correspond à un chemin de la racine à une feuille étiquetée avec 1. De manière duale, T peut être transformé en temps linéaire en une conjonction de clauses, notée $\mathtt{CNF}(T)$, où chaque clause est la négation du terme décrivant un chemin de la racine à une feuille étiquetée avec 0.

À titre d'illustration, l'exemple qui suit, utilisé dans la suite, illustre les concepts présentés dans ce chapitre :

Exemple 33. *L'arbre de décision représenté à la figure 24 sépare les orchidées Cattleya des autres orchidées en utilisant les attributs suivants : x_1 : "a des fleurs parfumées", x_2 : "a une ou deux feuilles", x_3 : "a de larges fleurs" et x_4 : "est sympodiale".*

Une représentation DNF *de T est donnée par :*

$$\mathtt{DNF}(T) = (\overline{x}_1 \wedge x_2 \wedge x_3 \wedge x_4) \vee (x_1 \wedge \overline{x}_2 \wedge \overline{x}_3 \wedge x_4) \vee (x_1 \wedge \overline{x}_2 \wedge x_3 \wedge x_4) \vee (x_1 \wedge x_2 \wedge \overline{x}_3 \wedge x_4) \vee (x_1 \wedge x_2 \wedge x_3 \wedge x_4)$$

De manière duale, une représentation CNF *de T est donnée par :*

$$\mathtt{CNF}(T) = (x_1 \vee x_2) \wedge (x_1 \vee \overline{x}_2 \vee x_3) \wedge (x_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee x_4) \wedge (\overline{x}_1 \vee x_2 \vee x_3 \vee x_4) \wedge (\overline{x}_1 \vee x_2 \vee \overline{x}_3 \vee x_4)$$

$$\wedge (\overline{x}_1 \vee \overline{x}_2 \vee x_3 \vee x_4) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee x_4).$$

6.1.2 Explications abductives

En XAI, les explications abductives visent à expliquer les raisons pour lesquelles un classifieur a pris une certaine décision.

Comme déjà indiqué, les arbres de décision fournissent une seule explication abductive explicite pour classer n'importe quelle instance d'entrée : sa *raison directe*. En général, cette raison diffère de l'instance elle-même (mais peut néanmoins coïncider avec elle dans certains cas).

Définition 54 (raison directe). *Soit $T \in \mathtt{DT}_d$ et $\boldsymbol{x} \in \{0,1\}^d$. La* raison directe *pour $\boldsymbol{x}$ donnée par T est le terme, noté $t_{\boldsymbol{x}}^T$, correspondant au chemin unique de la racine à une feuille de T qui est compatible avec $\boldsymbol{x}$, c'est-à-dire l'explication restreinte au chemin pour $\boldsymbol{x}$ donnée par T [IIMS20].*

Une autre notion importante d'explications abductives correspond au concept de raison suffisante.

Définition 55 (raison suffisante). *Soit $T \in \mathtt{DT}_d$ et $\boldsymbol{x} \in \{0,1\}^d$ tel que $T(\boldsymbol{x}) = 1$ (resp. $T(\boldsymbol{x}) = 0$). Une* raison suffisante *pour $\boldsymbol{x}$ étant donné T est un impliquant premier t de T (resp. $\neg T$) qui couvre $\boldsymbol{x}$. $sr(\boldsymbol{x}, T)$ désigne l'ensemble des raisons suffisantes pour $\boldsymbol{x}$ étant donné T.*

Ainsi, une raison suffisante [DH20] pour une instance $\boldsymbol{x}$ donnée d'une classe décrite par une fonction booléenne f est un sous-ensemble t des attributs de $\boldsymbol{x}$ qui est minimal par rapport à l'inclusion ensembliste, et tel que toute instance $\boldsymbol{x}'$ partageant cet ensemble t d'attributs soit classée par f de la même manière que $\boldsymbol{x}$. Ainsi, lorsque $f(\boldsymbol{x}) = 1$, t est une raison suffisante pour $\boldsymbol{x}$ donnée f si et seulement si t est un impliquant premier de f tel que t couvre $\boldsymbol{x}$, et lorsque $f(\boldsymbol{x}) = 0$, t est une raison suffisante pour $\boldsymbol{x}$ donnée f si et seulement si t est un impliquant premier de $\neg f$ tel que t couvre $\boldsymbol{x}$. En conséquence, les raisons suffisantes sont adaptées pour expliquer pourquoi l'instance $\boldsymbol{x}$ en question a été classée par f comme elle l'a été.

Contrairement aux raisons suffisantes qui sont des explications abductives minimales par rapport à l'inclusion ensembliste, les raisons directes peuvent contenir un nombre arbitraire d'attributs redondants, même dans le cas où f est représentée par un arbre de décision [IIMS20]. Cette redondance d'explication est fréquemment observée en pratique, même lorsque l'on considère des arbres de décision optimaux (de taille minimale). Lorsqu'on considère les raisons suffisantes d'une instance, on peut être intéressé à se concentrer sur les plus courtes d'entre elles, appelées raisons suffisantes de taille minimale (ou raisons suffisantes minimales). Ces raisons sont précieuses car la concision est souvent une propriété souhaitable des explications (selon le principe du rasoir d'Ockham). Formellement :

Définition 56 (raison suffisante minimale). *Soit $T \in \mathtt{DT}_d$ et $\boldsymbol{x} \in \{0,1\}^d$. Une* raison suffisante minimale *pour $\boldsymbol{x}$ étant donné T est une raison suffisante pour $\boldsymbol{x}$ étant donné T qui contient un nombre minimal de littéraux.*

Exemple 34. *Reprenons l'exemple 33. Nous pouvons observer que $T(\boldsymbol{x}) = 1$ pour l'instance $\boldsymbol{x} = (1,1,1,1)$ et que $T(\boldsymbol{x}') = 0$ pour l'instance $\boldsymbol{x}' = (0,0,0,0)$. La raison directe pour $\boldsymbol{x}$ étant donnée T est le terme $t^T_{\boldsymbol{x}} = x_1 \wedge x_2 \wedge x_3 \wedge x_4$. Il coïncide avec le terme $t_{\boldsymbol{x}}$ (instance complète).*

La raison directe pour $\boldsymbol{x}'$ étant donné T est le terme $t^T_{\boldsymbol{x}'} = \overline{x}_1 \wedge \overline{x}_2$, qui ne coïncide pas avec $t_{\boldsymbol{x}'} = \overline{x}_1 \wedge \overline{x}_2 \wedge \overline{x}_3 \wedge \overline{x}_4$. Les termes $x_1 \wedge x_4$ et $x_2 \wedge x_3 \wedge x_4$ sont des raisons suffisantes pour $\boldsymbol{x}$ étant donné T. $x_1 \wedge x_4$ est la seule raison suffisante minimale pour $\boldsymbol{x}$ étant donnée T. $\overline{x}_4$, $\overline{x}_1 \wedge \overline{x}_2$, et $\overline{x}_1 \wedge \overline{x}_3$ sont des raisons suffisantes pour $\boldsymbol{x}'$ étant donné T. $\overline{x}_4$ est la seule raison suffisante minimale pour $\boldsymbol{x}'$ étant donné T.

6.1.3 Explications contrastives

Contrairement aux raisons directes et aux raisons suffisantes (éventuellement de taille minimale) qui visent à expliquer la classification de l'instance $\boldsymbol{x}$ en essayant de répondre à la question pourquoi elle a été classée comme l'a fait le classifieur, les explications contrastives sont précieuses lorsque $\boldsymbol{x}$ n'a pas été classifiée comme prévu par l'utilisateur [Mil19]. Dans ce cas, on recherche des sous-ensembles minimaux d'attributs (par rapport à l'inclusion d'ensemble) qui, lorsqu'ils sont modifiés dans $\boldsymbol{x}$, suffisent à obtenir des instances qui sont classifiées positivement (ou négativement) par le classifieur f si $\boldsymbol{x}$ est classifiée négativement (ou positivement) par f. Notons que le calcul des explications contrastives peut également être utile lorsque $\boldsymbol{x}$ a été classifiée comme prévu par l'utilisateur. En effet, quelles que soient les attentes de l'utilisateur, les explications contrastives précisent comment modifier de manière minimale les instances pour obtenir une prédiction différente. En tant que telles, elles indiquent en quelque sorte dans quelle mesure la prédiction obtenue peut être considérée comme robuste [SCD18a, AKM20].

Définition 57 (explication contrastive). *Soit $T \in \mathtt{DT}_d$ et $\boldsymbol{x} \in \{0,1\}^d$ tel que $T(\boldsymbol{x}) = 1$ (resp. $T(\boldsymbol{x}) = 0$). Une explication contrastive pour $\boldsymbol{x}$ donnée T est un terme t sur X_d tel que $t \subseteq t_x$, $t_x \setminus t$ n'est pas un impliquant de T (resp. $\neg T$), et pour chaque $\ell \in t$, $t \setminus \ell$ ne satisfait pas cette dernière condition.*

Tout comme pour les raisons suffisantes, on peut être intéressé par se concentrer sur les explications contrastives les plus courtes :

Définition 58 (explication contrastive minimale). *Soit $T \in \mathtt{DT}_d$ et $\boldsymbol{x} \in \{0,1\}^d$. Une* explication contrastive minimale *pour $\boldsymbol{x}$ étant donné T est une explication contrastive pour $\boldsymbol{x}$ étant donné à T qui contient un nombre minimal de littéraux.*

Exemple 35. *En se basant sur l'exemple 33, nous pouvons observer que $T(\boldsymbol{x}) = 1$ pour l'instance $\boldsymbol{x} = (1,1,1,1)$ et que $T(\boldsymbol{x}') = 0$ pour l'instance $\boldsymbol{x}' = (0,0,0,0)$. Les termes x_4, $x_1 \wedge x_2$, et $x_1 \wedge x_3$ sont les explications contrastives pour $\boldsymbol{x}$ étant donné T. Ainsi, l'instance $(1,1,1,0)$ qui diffère de $\boldsymbol{x}$ uniquement sur x_4 n'est pas classifiée par T comme $\boldsymbol{x}$ l'est ($(1,1,1,0)$ est classifiée comme une instance négative). x_4 est la seule explication contrastive minimale pour $\boldsymbol{x}$ étant donné T. $\overline{x}_1 \wedge \overline{x}_4$ et $\overline{x}_2 \wedge \overline{x}_3 \wedge \overline{x}_4$ sont les explications contrastives pour $\boldsymbol{x}'$ étant donné T. Ainsi, l'instance $(1,0,0,1)$ qui diffère de $\boldsymbol{x}'$ uniquement sur x_1 et x_4 n'est pas classifiée par T comme $\boldsymbol{x}'$ l'est ($(1,0,0,1)$ est classifiée comme une instance positive). $\overline{x}_1 \wedge \overline{x}_4$ est la seule explication contrastive minimale pour $\boldsymbol{x}'$ étant donné T.*

Remarque 10. *Nous mentionnons en passant que, comme il s'agit d'arbres de décision T, nous aurions pu nous concentrer uniquement sur les explications pour les instances positives $\boldsymbol{x}$ étant donné T. Cela découle du fait que $\mathtt{DT}_d$ est fermé par négation, au sens où pour tout $T \in \mathtt{DT}_d$, un arbre de décision équivalent à $\neg T$ peut être obtenu en modifiant simplement l'étiquette de chaque feuille de T par son complément. Ainsi, pour toute instance $\boldsymbol{x} \in \{0,1\}^d$, une raison directe (resp. raison suffisante, raison suffisante minimale, explication contrastive) expliquant pourquoi $T(\boldsymbol{x}) = 0$ est précisément la même chose qu'une raison directe (resp. raison suffisante, raison suffisante de taille minimale, explication contrastive) expliquant pourquoi $(\neg T)(\boldsymbol{x}) = 1$. Considérer T ou sa négation $\neg T$ n'a aucun impact computationnel puisque $\neg T$ peut être calculé en temps linéaire à partir de T.*

6.2 Calculer toutes les explications abductives

Le nombre de raisons suffisantes peut être exponentiel. Lorsque l'on passe de la raison directe d'une instance (qui est unique mais, en général, pas exempte de redondances) à ses raisons suffisantes, un obstacle majeur à surmonter réside dans le nombre de raisons à considérer. En effet, même pour la classe restreinte des arbres de décision avec une profondeur logarithmique, une instance peut avoir un nombre exponentiel de raisons suffisantes : Nous avons démontré [ABB+22d] qu'il existe un arbre de décision $T \in \mathtt{DT}_d$ de profondeur $\log_2(d+1)$ tel que pour pour tout instance $\boldsymbol{x} \in \{0,1\}^d$, le nombre de raisons suffisantes pour $\boldsymbol{x}$ étant donné T est plus grand que $\lfloor \frac{3}{2}^{\frac{d+1}{2}} \rfloor$.

Dans de nombreux cas pratiques, le nombre de raisons suffisantes pour une instance étant donné un arbre de décision peut être très élevé. Ainsi, la figure 26 montre une instance `mnist49` (la sous-figure la plus à gauche) qui a été considérée dans nos expériences. Un arbre de décision de haute précision (voir la section 7.3) a été appris pour cet ensemble de données, et l'instance considérée a 569351040 raisons suffisantes étant donné cet arbre de décision.

Plusieurs algorithmes visant à générer l'ensemble de toutes les raisons suffisantes pour une instance donnée en fonction d'un arbre de décision ont été proposés jusqu'à présent (voir [IIMS20] pour un bref résumé). Ils se basent notamment sur la preuve que l'ensemble $sr(\boldsymbol{x}, T)$ de toutes les raisons suffisantes

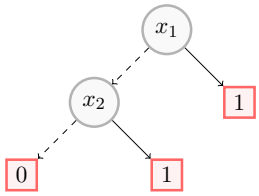


FIGURE 25 – Un arbre de décision T équivalent à $x_1 \vee x_2$.

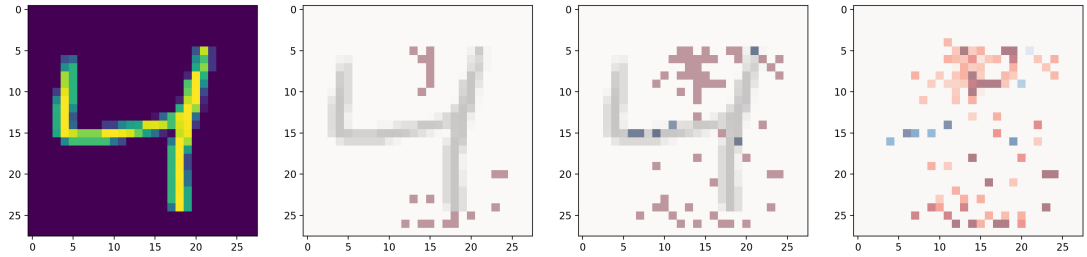


FIGURE 26 – Une instance `mnist49` (à gauche), puis (de gauche à droite) deux raisons suffisantes de cette instance, dont une de taille minimale (la plus à gauche des deux), et une carte de chaleur explicative pour l'instance (la figure la plus à droite). Les pixels formant l'instance « 4 » sont données en arrière plan mais ne font pas partie de la raison suffisante.

pour $\boldsymbol{x}$ étant donné T coïncide avec l'ensemble des impliquants premiers de la formule booléenne quantifiée

$$\forall\{\ell : \ell \in t_{\boldsymbol{x}}\} \cdot \texttt{CNF}(T)$$

où chaque littéral de $t_{\boldsymbol{x}}$ est universellement quantifié dans $\texttt{CNF}(T)$, en conjonction avec le fait qu'une formule `CNF` monotone équivalente à $\forall\{\ell : \ell \in t_{\boldsymbol{x}}\}.\texttt{CNF}(T)$ peut être obtenue en supprimant de chaque clause c de $\texttt{CNF}(T)$ chaque littéral n'appartenant pas à $t_{\boldsymbol{x}}$ (voir proposition 21 et théorème 10 de [DM21]). On peut exploiter un algorithme quasi-polynomial pour l'énumération incrémentale des impliquants premiers d'une formule `CNF` monotone afin de déduire $sr(\boldsymbol{x}, T)$ avec les mêmes garanties computationnelles. Cependant, il est à noter que l'existence d'un algorithme d'énumération pour les raisons suffisantes satisfaisant des exigences computationnelles strictes concernant le processus d'énumération (par exemple, temps polynomial incrémental, voire temps polynomial dans la taille cumulée de l'entrée et de la sortie) est peu vraisemblable [dCM22].

Une autre observation importante est que, dans le pire des cas, les raisons suffisantes pour une instance donnée peuvent différer sur chaque attribut. Par exemple, considérons l'arbre de décision T illustré à la figure 25. T est équivalent à $x_1 \vee x_2$. La raison directe pour $\boldsymbol{x} = (1, 1)$ étant donnée T est x_1, et c'est une raison suffisante pour $\boldsymbol{x}$. Cependant, $\boldsymbol{x}$ a également une autre raison suffisante, à savoir x_2. En pratique, les instances peuvent avoir des raisons suffisantes très différentes. À titre d'illustration, les deux sous-figures centrales de la figure 26 présentent deux raisons suffisantes pour la même instance, et on peut observer qu'elles diffèrent sur de nombreux attributs (les points bleus (resp. rouges) correspondent aux pixels allumés (resp. éteints)).

En résumé, d'une part, calculer l'ensemble de toutes les raisons suffisantes pour une instance donnée n'est pas toujours réalisable en un temps raisonnable. De plus, si le calcul réussit mais que la cardinalité de l'ensemble est énorme, l'ensemble des raisons suffisantes interprétés de manière disjonctive, équivalent à la *raison complète* de l'instance [DH20], est difficilement compréhensible par l'utilisateur.

D'autre part, en raison de la diversité qui peut être observée dans l'ensemble des raisons suffisantes, dériver une seule raison suffisante n'est pas nécessairement informatif. Ainsi, il est nécessaire de concevoir des approches pour synthétiser l'ensemble des raisons suffisantes tout en évitant les deux écueils (le problème computationnel et le problème informationnel).

6.2.1 Énumérer toutes les raisons suffisantes minimales

Une approche pour synthétiser de l'ensemble des raisons suffisantes consiste à se concentrer sur les raisons minimales. En effet, bien que l'ensemble des raisons suffisantes minimales pour une instance étant donné un arbre de décision puisse également être exponentiellement grand comme le montre la proposition 17, le nombre de raisons suffisantes minimales ne peut pas dépasser le nombre total de raisons suffisantes, et il peut être significativement plus faible en pratique.

Proposition 17. *Pour tout $d \in \mathbb{N}$ tel que d est impair, il existe un arbre de décision $T \in \mathtt{DT}_d$ de profondeur $\frac{d+1}{2}$ tel que T contient $2d+1$ nœuds et une instance $\boldsymbol{a} \in \{0,1\}^d$ tels que le nombre de raisons suffisantes minimales de $\boldsymbol{a}$ étant donné T est égal à $\sqrt{2}^{d-1}$.*

Avant de prouver ce résultat, nous introduisons d'abord la notation suivante. Soient E et F deux ensembles tels que $E \subseteq F$ et soit $\leq$ une relation de pré-ordre sur F.
$\max(E, \leq)$ est l'ensemble des éléments de E qui sont maximaux pour $\leq$, i.e., les éléments $x \in E$ tels qu'il n'existe pas $y \in E$ vérifiant $x \leq y$ et $y \not\leq x$ (de manière analogue, on définit $\min(E, \leq)$ comme l'ensemble des éléments de E qui sont minimaux pour $\leq$, i.e., les éléments de $x \in E$ tels qu'il n'existe pas $y \in E$ vérifiant $y \leq x$ et $x \not\leq y$).

Preuve 3. *Nous utilisons le lemme suivant (lemme 4) qui donne une caractérisation inductive de l'ensemble $sr(\boldsymbol{a}, f)$ des raisons suffisantes pour une instance $\boldsymbol{a}$ étant donné un classifieur booléen f :*

Lemme 4. *Pour toute fonction booléenne $f \in \mathcal{F}_d$ et toute instance $\boldsymbol{a} \in \{0,1\}^d$, $sr(\boldsymbol{a}, f)$ vérifie :*

$$
\begin{array}{ll}
sr(\boldsymbol{a}, 1) & = \{1\} \\
sr(\boldsymbol{a}, 0) & = \{\} \\
sr(\boldsymbol{a}, f) & = sr(\boldsymbol{a}, (f \mid \ell) \wedge (f \mid \overline{\ell})) \cup \{\ell \wedge t_\ell : t_\ell \in sr(\boldsymbol{a}, f \mid \ell) \text{ s.t. } t_\ell \not\models f \mid \overline{\ell}\} \\
 & \quad \text{où } Var(\ell) \subseteq Var(f) \text{ et } t_{\boldsymbol{a}} \models \ell
\end{array}
$$

et

$$sr(\boldsymbol{a}, (f \mid \ell) \wedge (f \mid \overline{\ell})) = max(\{t_\ell \wedge t_{\overline{\ell}} : t_\ell \in sr(\boldsymbol{a}, f \mid \ell), t_{\overline{\ell}} \in sr(\boldsymbol{a}, f \mid \overline{\ell})\}, \models).$$

Démonstration. Rappelons d'abord la caractérisation inductive suivante de $pi(f)$, l'ensemble des implicants premiers de $f \in \mathcal{F}_d$, basé sur la décomposition de Shannon de f donné par $(\overline{x} \wedge f(\overline{x})) \vee (x \wedge f(x))$ (voir [BHMSV84]) :

$$
\begin{array}{ll}
pi(1) & = \{1\} \\
pi(0) & = \{\} \\
pi(f) & = pi((f \mid \overline{x}) \wedge (f \mid x)) \\
 & \quad \cup \{\overline{x} \wedge t_{\overline{x}} : t_{\overline{x}} \in pi(f \mid \overline{x}) \text{ t.q. } \nexists t \in pi((f \mid \overline{x}) \wedge (f \mid x)), t_{\overline{x}} \models t\} \\
 & \quad \cup \{x \wedge t_x : t_x \in pi(f \mid x) \text{ t.q. } \nexists t \in pi((f \mid \overline{x}) \wedge (f \mid x)), t_x \models t\} \\
 & \quad \text{où } x \in Var(f)
\end{array}
$$

et

$$pi((f \mid \overline{x}) \wedge (f \mid x)) = max(\{t_{\overline{x}} \wedge t_x : t_{\overline{x}} \in pi(f \mid \overline{x}), t_x \in pi(f \mid x)\}, \models).$$

Pour les cas de base $sr(\boldsymbol{a}, 1) = \{1\}$ et $sr(\boldsymbol{a}, 0) = \{\}$, le résultat est évident. Pour le cas général, nous avons :

$$\begin{aligned} sr(\boldsymbol{a}, f) &= \{t \in pi(f) : t_{\boldsymbol{a}} \models t\} \\ &= \{t \in pi((f \mid \overline{x}) \wedge (f \mid x)) : t_{\boldsymbol{a}} \models t\} \\ &\cup \{\overline{x} \wedge t_{\overline{x}} : t_{\overline{x}} \in pi(f \mid \overline{x}) \text{ t.q. } (\nexists t' \in pi((f \mid \overline{x}) \wedge (f \mid x)), t_{\overline{x}} \models t') \text{ et } t_{\boldsymbol{a}} \models \overline{x} \wedge t_{\overline{x}}\} \\ &\cup \{x \wedge t_x : t_x \in pi(f \mid x) \text{ t.q. } (\nexists t' \in pi((f \mid \overline{x}) \wedge (f \mid x)), t_x \models t') \text{ et } t_{\boldsymbol{a}} \models x \wedge t_x\} \\ &= sr(\boldsymbol{a}, (f \mid \overline{x}) \wedge (f \mid x)) \\ &\cup \{\overline{x} \wedge t_{\overline{x}} : t_{\overline{x}} \in pi(f \mid \overline{x}) \text{ t.q. } (\nexists t' \in pi((f \mid \overline{x}) \wedge (f \mid x)), t_{\overline{x}} \models t') \text{ et } t_{\boldsymbol{a}} \models \overline{x} \wedge t_{\overline{x}}\} \\ &\cup \{x \wedge t_x : t_x \in pi(f \mid x) \text{ t.q. } (\nexists t' \in pi((f \mid \overline{x}) \wedge (f \mid x)), t_x \models t') \text{ et } t_{\boldsymbol{a}} \models x \wedge t_x\} \end{aligned}$$

Maintenant, puisque x est une instance, on a soit $t_x \models x$, soit $t_x \models \overline{x}$.

Supposons que $t_x \models x$ (le cas $t_x \models \overline{x}$ est similaire). Dans cette situation, l'ensemble $\{\overline{x} \wedge t_{\overline{x}} : t_{\overline{x}} \in pi(f \mid \overline{x})$ t.q. $(\nexists t \in pi((f \mid \overline{x}) \wedge (f \mid x)), t_{\overline{x}} \models t)$ et $t_{\boldsymbol{a}} \models \overline{x} \wedge t_{\overline{x}}\}$ est vide. En conséquence, nous obtenons :

$$\begin{aligned} sr(\boldsymbol{a}, f) &= sr(\boldsymbol{a}, (f \mid \overline{x}) \wedge (f \mid x)) \\ &\cup \{\ell \wedge t_\ell : t_\ell \in pi(f \mid \ell) \text{ t.q. } (\nexists t' \in pi((f \mid \overline{\ell}) \wedge (f \mid \ell)), t_\ell \models t') \text{ et } t_{\boldsymbol{a}} \models \ell \wedge t_\ell\} \end{aligned}$$

Considérons maintenant la condition $\exists t' \in pi((f \mid \overline{\ell}) \wedge (f \mid \ell)), t_\ell \models t'$ et supposons qu'elle soit satisfaite. Puisque $pi((f \mid \overline{\ell}) \wedge (f \mid \ell)) = max(\{t'_{\overline{\ell}} \wedge t'_\ell : t'_{\overline{\ell}} \in pi(f \mid \overline{\ell}), t'_\ell \in pi(f \mid \ell)\}, \models)$, il existe $t'_{\overline{\ell}} \in pi(f \mid \overline{\ell})$ et $t'_\ell \in pi(f \mid \ell)$ tels que $t' \equiv t'_{\overline{\ell}} \wedge t'_\ell$. Ainsi, on a $t_\ell \models t'_{\overline{\ell}} \wedge t'_\ell$, et en particulier $t_\ell \models t'_\ell$. Mais comme t_ℓ et t'_ℓ sont des impliquants premiers de $f \mid \ell$, cela implique que $t_\ell \equiv t'_\ell$. En outre, de $t_\ell \models t'_{\overline{\ell}} \wedge t'_\ell$ on dérive que $t_\ell \models t'_{\overline{\ell}}$. Enfin, un impliquant premier $t'_{\overline{\ell}}$ de $f \mid \overline{\ell}$ tel que $t_\ell \models t'_{\overline{\ell}}$ existe si seulement si $t_\ell \models f \mid \overline{\ell}$. Donc, la condition $(\exists t' \in pi((f \mid \overline{\ell}) \wedge (f \mid \ell)), t_\ell \models t')$ est équivalente à $t_\ell \models f \mid \overline{\ell}$. Ainsi, nous obtenons :

$$\begin{aligned} sr(\boldsymbol{a}, f) &= sr(\boldsymbol{a}, (f \mid \ell) \wedge (f \mid \overline{\ell})) \\ &\cup \{\ell \wedge t_\ell : t_\ell \in sr(\boldsymbol{a}, f \mid \ell) \text{ t.q. } t_\ell \not\models f \mid \overline{\ell}\} \end{aligned}$$

Finalement, montrons que :

$sr(x, f(\ell) \wedge f(\overline{\ell})) = max(\{t_\ell \wedge t_{\overline{\ell}} : t_\ell \in sr(x, f \mid \ell), t_{\overline{\ell}} \in sr(x, f \mid \overline{\ell})\}, \models)$
On sait que $sr(x, (f \mid \ell) \wedge (f \mid \overline{\ell})) = \{t \in pi((f \mid \ell) \wedge (f \mid \overline{\ell})) : t_x \models t\}$

Or, $pi((f \mid \ell) \wedge (f \mid \overline{\ell})) = max(\{t_\ell \wedge t_{\overline{\ell}} : t_\ell \in pi(f \mid \ell), t_{\overline{\ell}} \in pi(f \mid \overline{\ell})\}, \models)$

Soit $t \in pi((f \mid \ell) \wedge (f \mid \overline{\ell}))$. Alors il existe $t_\ell \in pi(f \mid \ell)$ et il existe $t_{\overline{\ell}} \in pi(f \mid \overline{\ell})$ tel que $t \equiv t_\ell \wedge t_{\overline{\ell}}$.

On a $t_x \models t$ si et seulement et si $t_x \models t_\ell$ et $t_x \models t_{\overline{\ell}}$

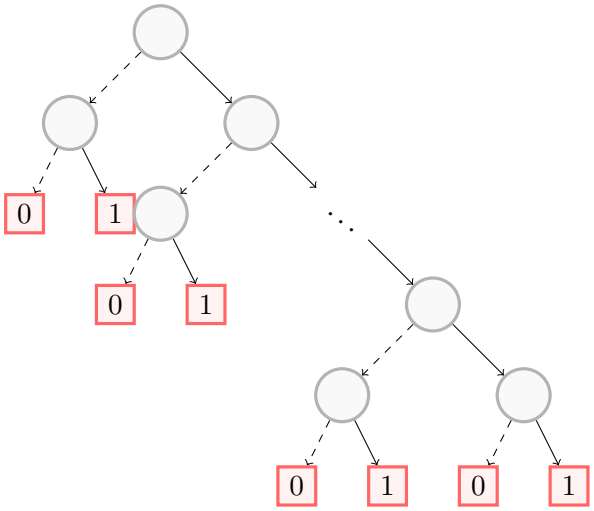
Donc $\{t \in pi((f \mid \ell) \wedge (f \mid \overline{\ell})) : t_x \models t\} = max(\{t_\ell \wedge t_{\overline{\ell}} : t_\ell \in pi(f \mid \ell), t_{\overline{\ell}} \in pi(f \mid \overline{\ell}), t_x \models t_\ell, t_x \models t_{\overline{\ell}}\}, \models) = max(\{t_\ell \wedge t_{\overline{\ell}} : t_\ell \in sr(\boldsymbol{a}, f \mid \ell), t_{\overline{\ell}} \in sr(\boldsymbol{a}, f \mid \overline{\ell}), \models)$,

ce qui conclut la preuve du lemme 4.

□

De la caractérisation inductive de $sr(\boldsymbol{a}, f)$ donnée par la lemme précédent, on peut facilement dériver un algorithme ascendant « bottom-up » permettant de calculer $sr(\boldsymbol{a}, f)$ lorsque f est un arbre de décision.

Considérons maintenant un arbre de décision T de profondeur $k \geq 1$ ayant la forme suivante :



T a $2k-1$ nœuds de décision et $2k$ feuilles. Supposons que les variables associées aux nœuds de décision sont en correspondance bijective avec les nœuds de décision (i.e., les variables sont toutes distinctes). Le nombre de variables dans T est ainsi $d = 2k-1$. Considérons maintenant l'instance $\boldsymbol{a} \in \{0,1\}^d$ tel que $x_i = 1$ pour chaque $i \in [d]$. Nous allons prouver par induction sur la profondeur k de T que l'instance $\boldsymbol{a}$ a 2^{k-1} raisons minimales étant donné T et chaque raison minimale donnée par le lemme 4 contient k littéraux. La preuve tire avantage de la caractérisation récursive de l'ensemble de toutes les raisons suffisantes pour une instance étant donné un arbre de décision, comme précisé dans le lemme 4.

- *Le cas de base $k = 1$. On a $d = 1$. T se résume en un nœud de décision étiqueté par la variable unique x de X_d, un enfant gauche qui est une 0-feuille et son enfant droit est une 1-feuille. T est équivalent à x et x est impliqué par $t_{\boldsymbol{a}}$. D'où x est l'unique raison suffisante pour $\boldsymbol{a}$ étant donné T, donc l'unique raison minimale pour $\boldsymbol{a}$ étant donné T. Comme prévu, le nombre de raisons minimales de $\boldsymbol{a}$ étant donné T est égal à 2^{k-1}. La taille de l'unique raison minimale est $k = 1$.*
- *L'étape inductive $k > 1$. Soit x une variable de X_d étiquetant le nœud racine de T. Par construction, l'enfant gauche T_l de T est équivalent à une seule variable x_l, qui est l'unique raison minimale pour $\boldsymbol{a}$ étant donné T_l. L'enfant droit T_r de T a la même forme que T, mais avec profondeur $k-1$. Par induction, on sait que $\boldsymbol{a}$ a 2^{k-2} raisons minimales étant donné T_r, chaque raisons contient $k-1$ littéraux. Comme le montre le lemme 4, à condition que les variables étiquetant les nœuds de décision soient deux à deux distincts, les raisons minimales de $\boldsymbol{a}$ étant donné T sont obtenues en étendant chaque raison minimale pour $\boldsymbol{a}$ étant donné T_r avec x_l et en étendant chaque raison minimale pour $\boldsymbol{a}$ étant donné T_r avec x. Par conséquent, $\boldsymbol{a}$ a $2 \times (2^{k-2}) = 2^{k-1}$ raisons minimales étant donné T et chaque raison contient $k-1+1 = k$ littéraux.*

Finalement, puisque $d = 2k-1$, on a $k = \frac{d+1}{2}$ et le nombre de raisons minimales pour $\boldsymbol{a}$ étant donné T est égal à $2^{k-1} = \sqrt{2}^{d-1}$.

Il est connu qu'une raison suffisante pour une instance $\boldsymbol{a}$ étant donné un arbre de décision T peut être générée en temps polynomial par rapport à la taille de l'entrée ($\boldsymbol{a}$ et T) en utilisant un algorithme glouton (voir par exemple, [IIMS20]). L'ajout de la contrainte de taille minimale rend le problème intraitable :

Proposition 18. *[BMPS20, ABB+22d] Soit $T \in \mathtt{DT}_d$ et $\boldsymbol{a} \in \{0,1\}^d$. Calculer une raison suffisante minimale pour $\boldsymbol{a}$ étant donné T est* NP*-difficile.*

Malgré ce résultat d'intraitabilité, les raisons suffisantes minimales peuvent être générées dans de nombreux cas pratiques. Une approche courante pour traiter les problèmes d'optimisation NP consiste à s'appuyer sur des solveurs de contraintes. On suit cette démarche ici et on formule la tâche de trouver des raisons suffisantes minimales comme un problème d'optimisation sous contraintes. Rappelons qu'un problème PARTIAL MAXSAT consiste en une paire $(C_{\text{soft}}, C_{\text{hard}})$ où C_{soft} et C_{hard} sont des ensembles (finis) de clauses. L'objectif est de déterminer une affectation booléenne qui maximise le nombre de clauses c dans C_{soft} qui sont satisfaites, tout en satisfaisant toutes les clauses de C_{hard}.

Remarque 11. *Pour la bonne compréhension de la proposition 19 et de sa preuve, rappelons que pour une instance $\boldsymbol{a} \in \{0,1\}^d$, le terme $t_{\boldsymbol{a}}$ est l'ensemble des littéraux sur $X_d = \{x_1, \cdots, x_d\}$ correspondant à l'instance $\boldsymbol{a}$. En d'autres termes, pour tout $i \in [d]$, le littéral x_i (resp. $\overline{x_i}$) appartient à $t_{\boldsymbol{a}}$ si et seulement si la i-ème coordonnée de $\boldsymbol{a}$ vaut 1 (resp. 0). Les clauses $c \in \texttt{CNF}(T)$ sont également vues comme des ensembles de littéraux sur $\{x_1, \cdots, x_d\}$.*

Proposition 19. *Soit T un arbre de décision dans $\texttt{DT}_d$ et $\boldsymbol{a} \in \{0,1\}^d$ une instance telle que $T(\boldsymbol{a}) = 1$. Soit $(C_{\text{soft}}, C_{\text{hard}})$ une instance du problème* PARTIAL MAXSAT *telle que :*

$$C_{\text{soft}} = \{\overline{x}_i : a_i = 1, i \in [d]\} \cup \{x_i : \overline{a}_i = 0, i \in [d]\} \text{ and } C_{\text{hard}} = \{c \cap t_{\boldsymbol{a}} : c \in \texttt{CNF}(T)\}.$$

L'intersection de $t_{\boldsymbol{a}}$ avec $t_{\boldsymbol{x}^*}$ où $\boldsymbol{x}^*$ est une solution optimale de $(C_{\text{hard}}, C_{\text{soft}})$ est une raison suffisante minimale pour $\boldsymbol{a}$ étant donné T.

Si $\boldsymbol{a}$ est tel que $T(\boldsymbol{a}) = 0$, alors il suffit de considérer la même instance de PARTIAL MAXSAT que celle mentionnée précédemment, à ceci près que $C_{\text{hard}} = \{c \cap t_{\boldsymbol{a}} : c \in \texttt{CNF}(\neg T)\}$.

Preuve 4. *Soit $\boldsymbol{x}^*$ une solution de $(C_{\text{soft}}, C_{\text{hard}})$. On observe que l'ensemble de toutes les clauses dures $c \cap t_{\boldsymbol{a}}$ (où c est une clause de $\texttt{CNF}(T)$) correspond à une formule* CNF monotone *dans laquelle seuls les littéraux de $t_{\boldsymbol{a}}$ apparaissent. Pour tout modèle $t_{\boldsymbol{x}^*}$ de C_{hard}, $t_{\boldsymbol{x}^*} \cap t_{\boldsymbol{a}}$ constitue donc un impliquant t (qui par construction est impliqué par $t_{\boldsymbol{a}}$).*

Les clauses soft de C_{soft} sont utilisées pour sélectionner parmi les affectations qui satisfont toutes les clauses dures, celles qui correspondent à des raisons suffisantes minimales. Les clauses soft sont données par les littéraux ℓ_i, qui sont précisément les littéraux complémentaires de ceux apparaissant dans $t_{\boldsymbol{a}}$. Avoir une clause soft ℓ_i violée par $\boldsymbol{x}^*$ signifie que le littéral $\overline{\ell}$ de $t_{\boldsymbol{a}}$ est nécessaire pour obtenir un impliquant de T étant donné l'affectation des autres variables dans $\boldsymbol{x}^*$. Chaque fois qu'une clause soft ℓ_i est violée par $\boldsymbol{x}^*$ une pénalité de 1 est imposé. Cela assure que le terme composé des littéraux qui sont partagés par $t_{\boldsymbol{a}}$ et $t_{\boldsymbol{x}^*}$ est une raison suffisante minimale pour $\boldsymbol{a}$ étant donné T.

Exemple 36. *Considérons l'arbre de décision T donné à la figure 24. $\texttt{CNF}(T)$ est donnée à l'exemple 33. Pour l'instance $\boldsymbol{a} = (1,1,1,1)$, l'ensemble $\{c \cap t_{\boldsymbol{a}} : c \in \texttt{CNF}(T)\}$ vaut $\{x_1 \vee x_2, x_1 \vee x_3, x_1 \vee x_4, x_2 \vee x_3 \vee x_4, x_2 \vee x_4, x_3 \vee x_4, x_4\}$ et est équivalent à $\{x_1 \vee x_2, x_1 \vee x_3, x_4\}$ que l'on peut donc retenir comme ensemble de contraintes dures. L'ensemble de contraintes soft est $\{\overline{x_1}, \overline{x_2}, \overline{x_3}, \overline{x_4}\}$. L'instance* PARTIAL MAXSAT *obtenue possède une solution optimale unique $\boldsymbol{x}^* = (1,0,0,1)$. Donc l'instance $\boldsymbol{a}$ possède une raison suffisante minimale unique étant donné T, à savoir $t_{\boldsymbol{a}} \cap t_{\boldsymbol{x}^*} = \{x_1, x_4\}$.*

On peut noter qu'il est possible de tirer parti de la caractérisation de PARTIAL MAXSAT ci-dessus pour générer un nombre prédéfini de raisons suffisantes minimales en utilisant des clauses de blocage. Fondamentalement, l'approche est la suivante : on génère une première raison t, puis on ajoute à C_{hard} la négation de t en tant que clause et on continue jusqu'à ce que le nombre prédéfini de raisons soit atteint ou qu'aucune solution n'existe ou que la taille de la dernière solution générée soit strictement supérieure à la taille de la première raison t qui a été calculée. Pour éviter le dernier test, après la génération de la

première raison t, on peut alternativement ajouter à C_{hard} un encodage CNF d'une contrainte de cardinalité garantissant que les raisons suivantes à générer ont la même taille que celle de t.

Enfin, lorsqu'il s'agit d'instances $\boldsymbol{a}$ pour lesquelles le calcul d'une raison suffisante de taille minimale t est trop exigeant en pratique (c'est-à-dire que le solveur PARTIAL MAXSAT utilisé pour le calcul ne termine pas au bout d'un temps raisonnable), on peut assouplir la condition de minimalité de la taille des explications et calculer une explication abductive h pour $\boldsymbol{a}$ étant donné T en temps polynomial, tout en garantissant certaines propriétés quant à la taille de h. En effet, la dérivation d'une raison de taille minimale t pour une instance $\boldsymbol{a}$ étant donné un arbre de décision T revient au calcul d'un ensemble de recouvrement minimal t de l'hypergraphe ayant $\{c \cap t_{\boldsymbol{a}} : c \in \texttt{CNF}(T)\}$ comme ensemble d'hyperarêtes. Le point clé est qu'un algorithme glouton simple qui s'exécute en temps $\mathcal{O}(d \cdot |T|)$ peut être utilisé pour dériver un ensemble de recouvrement h d'un tel hypergraphe : cet algorithme consiste à choisir un littéral ℓ de $t_{\boldsymbol{a}}$ qui appartient au plus grand nombre de clauses de $\{c \cap t_{\boldsymbol{a}} : c \in \texttt{CNF}(T)\}$, puis à supprimer ℓ de $t_{\boldsymbol{a}}$ et toutes les clauses contenant ℓ de $\{c \cap t_{\boldsymbol{a}} : c \in \texttt{CNF}(T)\}$, et à répéter ce processus jusqu'à ce que l'ensemble de clauses résultant soit vide. h est l'ensemble de littéraux ℓ de $t_{\boldsymbol{a}}$ qui ont été sélectionnés dans le processus. Il s'avère que la taille de h est garantie d'être inférieure à la taille de n'importe quelle raison suffisante minimale t un facteur $\ln m - \ln\ln m + 0.78$ prés [Sla97], où m est le nombre de clauses dans $\texttt{CNF}(T)$ (ou, de manière équivalente, la taille de T (notée $|T|$).

6.2.2 Synthétiser l'ensemble des raisons suffisantes

Synthétiser l'ensemble des raisons suffisantes. Nous introduisons maintenant les notions d'attributs nécessaires / pertinents pour pallier ces problèmes. Ces notions font écho à celles qui ont été mises en avant dans [EG95] pour l'abduction logique.

Définition 59 (Attributs explicatifs). *Soit $f \in \mathcal{F}_d$ et $\boldsymbol{x} \in \{0,1\}^d$ une instance. Soit e un type d'explication (abductif ou contrastif).* [13]

- *Un littéral ℓ sur X_d est un* attribut nécessaire *pour la famille d'explications e pour $\boldsymbol{x}$ étant donné f si et seulement si ℓ appartient à chaque explication t pour $\boldsymbol{x}$ étant donné f telle que t soit de type e. $Nec_e(\boldsymbol{x}, f)$ désigne l'ensemble de tous les attributs nécessaires pour la famille d'explications e pour $\boldsymbol{x}$ étant donné f.*
- *Un littéral ℓ sur X_d est un* attribut pertinent *pour la famille d'explications e pour $\boldsymbol{x}$ étant donné f si et seulement si ℓ appartient à au moins une explication t pour $\boldsymbol{x}$ étant donné f telle que t soit de type e. $Rel_e(\boldsymbol{x}, f)$ désigne l'ensemble de tous les attributs pertinents pour la famille e d'explications pour $\boldsymbol{x}$ étant donné f. $Irr_e(\boldsymbol{x}, f)$, qui est le complémentaire de $Rel_e(\boldsymbol{x}, f)$ dans l'ensemble de tous les littéraux sur X_d, désigne l'ensemble de tous les attributs non pertinents pour la famille d'explications e pour $\boldsymbol{x}$ étant donné f.*

Les attributs nécessaires (respectivement non pertinents) pour la famille s des raisons suffisantes pour $\boldsymbol{x}$ étant donnée f sont les attributs les plus (respectivement moins) importantes pour expliquer la classification de $\boldsymbol{x}$ par f, car ils appartiennent à chaque raison suffisante (respectivement à aucune) pour $\boldsymbol{x}$ étant donnée f. Lorsqu'une seule raison suffisante t pour $\boldsymbol{x}$ étant donné f a été calculée, la cardinalité de t privé des attributs de $Nec_s(\boldsymbol{x}, f)$ est petite, et la cardinalité de la différence symétrique entre t et $Rel_s(\boldsymbol{x}, f)$ est également petite, t peut être considéré comme un bon représentant de la *raison complète* de $\boldsymbol{x}$ étant donné f, dans le sens qu'une autre raison suffisante t' pour $\boldsymbol{x}$ étant donné f qui diffère beaucoup de t ne peut pas exister.

13. Par exemple, e peut être s lorsque les raisons suffisantes pour $\boldsymbol{x}$ étant donné f sont ciblées ou c lorsque les explications contrastives pour $\boldsymbol{x}$ étant donné f sont ciblées.

Dans le cas où f est un arbre de décision T, bien que l'ensemble de toutes les raisons suffisantes pour $\boldsymbol{x}$ étant donné T ne puisse pas être généré s'il est trop grand, $Nec_s(\boldsymbol{x}, T)$, $Rel_s(\boldsymbol{x}, T)$ et $Irr_s(\boldsymbol{x}, T)$ peuvent être déduits efficacement :

Proposition 20 ([ABB+22d]). *Soit $T \in \mathtt{DT}_d$, et $\boldsymbol{x} \in \{0,1\}^d$. Calculer $Nec_s(\boldsymbol{x}, T)$, $Rel_s(\boldsymbol{x}, T)$, et $Irr_s(\boldsymbol{x}, T)$ peut se faire en temps $\mathcal{O}((d + |T|) \cdot |T|)$.*

Avant de commencer la preuve de la proposition 20, il est important de rappeler la notion de raison complète qui sera utilisée dans la preuve.

Définition 60 (**raison complète** [DH20]). *La raison complète pour $\boldsymbol{x}$ étant donné f est la disjonction de toutes ses raisons suffisantes.*

Il s'agit donc d'une formule propositionnelle en DNF, mais comme toute formule propositionnelle, on peut l'écrire sous forme équivalente dans divers formats (dont le format CNF).

Preuve 5. *Pour calculer $Nec_s(\boldsymbol{x}, T)$, $Rel_s(\boldsymbol{x}, T)$ et $Irr_s(\boldsymbol{x}, T)$, on procède comme suit. Tout d'abord, calculons $\mathtt{CNF}(T)$, puis retirons de cet ensemble de clauses chaque littéral qui n'appartient pas à $t_{\boldsymbol{x}}$. Cela peut être fait en temps $\mathcal{O}(d \cdot |T|)$. Par construction, la formule CNF résultante (disons f) est monotone : chaque littéral qu'elle contient a la même polarité que celle qu'il a dans $t_{\boldsymbol{x}}$. De plus, la taille de f ne peut pas dépasser la taille de $\mathtt{CNF}(T)$, donc la taille de T. D'après [DM21], f équivaut à la raison complète pour $\boldsymbol{x}$ étant donné T, c'est-à-dire à la disjonction de toutes les raisons suffisantes pour $\boldsymbol{x}$ étant donné T.*

Étant donné que f est une formule CNF monotone, ses impliqués premiers peuvent être calculés en supprimant de f chaque clause qui est un sur-ensemble strict d'une autre clause de f. Cela peut être réalisé en temps quadratique par rapport à la taille de f, donc par rapport à la taille de T. Soit g la formule CNF résultante donné par ces impliqués premiers et équivalente à f. g équivaut à f donc g est équivalente à la raison complète pour $\boldsymbol{x}$ étant donné T. Étant donné qu'elle est sous forme d'impliqués premiers, g dépend de chaque littéral qui y apparaît (voir la proposition 8 dans [LLM03] pour plus de détails), donc il en va de même pour la raison complète pour $\boldsymbol{x}$ étant donné T : les littéraux dont elle dépend sont ceux qui apparaissent dans g.

Cela signifie que pour chaque littéral ℓ apparaissant dans g, il existe une raison suffisante pour $\boldsymbol{x}$ selon T qui contient ℓ, de sorte que $Rel_s(\boldsymbol{x}, T)$ est l'ensemble des littéraux apparaissant dans g et $Irr_s(\boldsymbol{x}, T)$ est le complément de $Rel_s(\boldsymbol{x}, T)$ dans l'ensemble de tous les littéraux sur X_d. Enfin, puisque par définition les littéraux de $Nec_s(\boldsymbol{x}, T)$ doivent appartenir à chaque raison suffisante pour $\boldsymbol{x}$ étant donné T, ils sont donnés par les clauses unitaires qui appartiennent à g.

Exemple 37. *On a vu à l'exemple 36 précédent que pour l'arbre de décision donné à la figure 24 et pour l'instance $\boldsymbol{a} = (1,1,1,1)$, on a :*

$$\{c \cap t_{\boldsymbol{a}} : c \in \mathtt{CNF}(T)\} = \{x_1 \vee x_2, x_1 \vee x_3, x_1 \vee x_4, x_2 \vee x_3 \vee x_4, x_2 \vee x_4, x_3 \vee x_4, x_4\}$$

En supprimant de cette CNF monotone les clauses sous-sommées, on obtient les trois clauses $x_1 \vee x_2, x_1 \vee x_3, x_4$ qui sont les impliqués premiers de cette CNF, donc de la raison complète pour $\boldsymbol{a}$ étant donné T. Et comme les littéraux x_1, x_2, x_3, x_4 apparaissent dans la CNF $\{x_1 \vee x_2, x_1 \vee x_3, x_4\}$, la raison complète pour $\boldsymbol{a}$ étant donné T dépend exactement de ces littéraux :

$$Rel_s(\boldsymbol{a}, T) = \{x_1, x_2, x_3, x_4\} \text{ et } Irr_s(\boldsymbol{a}, T) = \{\overline{x_1}, \overline{x_2}, \overline{x_3}, \overline{x_4}\}$$

Les littéraux nécessaires sont ceux qui apparaissent dans chaque impliquant premier de la raison complète pour $\boldsymbol{a}$ étant donné T, donc ils correspondent aux clauses unitaires de CNF, dans notre exemple $Nec_s(\boldsymbol{a}, T) = \{x_4\}$.

Nous mentionnons en passant que la tâche de décider si un attribut donné appartient à une raison suffisante pour une instance d'entrée est également appelée le *problème d'appartenance d'attributs* [HIIMS21]. Bien que ce problème soit NP-complet pour des classifieurs booléens arbitraires, il reste traitable pour les arbres de décision [HIIMS21]. Ce résultat de faisabilité est en parfait accord avec notre proposition 20.

Aller plus loin consiste à évaluer le pouvoir explicatif de chaque attribut (positif ou négatif) :

Définition 61 (Pouvoir explicatif). *Soit $f \in \mathcal{F}_d$, et $\boldsymbol{x} \in \{0,1\}^d$ une instance. Soit e un type d'explication, et $E_e(\boldsymbol{x}, f)$ l'ensemble de toutes les explications de $\boldsymbol{x}$ étant donné f qui sont de type e. Le* pouvoir explicatif *d'un littéral ℓ sur X_d pour $\boldsymbol{x}$ étant donné f w.r.t. e est donné par*

$$Imp_e(\ell, \boldsymbol{x}, f) = \frac{\#(\{t \in E_e(\boldsymbol{x}, f) : \ell \in t\})}{\#(E_e(\boldsymbol{x}, f))}.$$

Exemple 38. *Revenons à l'exemple 34. Nous avons $Imp_s(x_4, \boldsymbol{x}, T) = 1$, $Imp_s(x_1, \boldsymbol{x}, T) = Imp_s(x_2, \boldsymbol{x}, T) = Imp_s(x_3, \boldsymbol{x}, T) = \frac{1}{2}$, et $Imp_s(\ell, \boldsymbol{x}, T) = 0$ pour tout autre littéral ℓ (les littéraux négatifs sur $\{x_1, x_2, x_3, x_4\}$).* [14] *En ce qui concerne $\boldsymbol{x}'$, nous avons $Nec_s(\boldsymbol{x}', T) = \varnothing$ et $Rel_s(\boldsymbol{x}', T) = \{\overline{x}_1, \overline{x}_2, \overline{x}_3, \overline{x}_4\}$. De plus, nous avons $Imp_s(\overline{x}_1, \boldsymbol{x}', T) = \frac{2}{3}$, $Imp_s(\overline{x}_2, \boldsymbol{x}', T) = Imp_s(\overline{x}_3, \boldsymbol{x}', T) = Imp_s(\overline{x}_4, \boldsymbol{x}', T) = \frac{1}{3}$, et $Imp_s(\ell, \boldsymbol{x}', T) = 0$ pour tout autre littéral ℓ (les littéraux positifs sur $\{x_1, x_2, x_3, x_4\}$).*

Il est important de ne pas confondre le concept de pouvoir explicatif avec la notion d'importance des attributs (qui peut être définie et évaluée de différentes manières) : le premier est local (c'est-à-dire relatif à une instance) et non global, il concerne les littéraux (i.e. les caractéristiques booléennes) et non les variables (la polarité compte), et il est lié à la tâche d'explication, et non à la prédiction. Pour calculer le pouvoir explicatif d'un littéral, une approche directe consiste à énumérer les explications de $E_e(\boldsymbol{x}, f)$. Ceci est réalisable lorsque cet ensemble n'est pas trop grand, mais, comme déjà mentionné, ce n'est pas toujours le cas pour les raisons suffisantes, même lorsque f est un arbre de décision T. Ainsi, pour traiter le cas restant, il faut définir une approche alternative.

En se concentrant sur les raisons suffisantes, nous avons proposé une telle approche pour calculer le pouvoir explicatif $Imp_s(\ell, \boldsymbol{x}, T)$ par rapport aux raisons suffisantes d'un littéral ℓ pour une instance $\boldsymbol{x}$ étant donné un arbre de décision T. Cette approche est présentée dans l'algorithme 4. Comme expliqué précédemment, nous savons que $sr(\boldsymbol{x}, T)$ est, par construction, l'ensemble des impliquants premiers de la formule CNF, $C = \{c \cap t_{\boldsymbol{x}} : c \in \texttt{CNF}(T)\}$, qui peut être calculée en temps polynomial par rapport à la taille de T et la taille de $\boldsymbol{x}$. Ensuite, on peut exploiter le processus de traduction présenté dans [JMSSS14] qui montre comment associer en temps polynomial à une formule CNF donnée (ici, C) une autre formule CNF, appelée PIC, telle que les modèles de PIC correspondent bijectivement aux impliquants premiers de C. La traduction s'appuie sur la transformation de Tseitin [Tse68], et nécessite l'introduction de variables auxiliaires. Chaque variable auxiliaire introduite est définie à partir des variables initiales [LM08], de sorte que le nombre de modèles de la formule CNF résultante PIC est le même que le nombre d'impliquants premiers de C. Dans notre cas, la traduction peut être simplifiée car C est une formule CNF monotone. Enfin, nous tirons parti du compilateur D4 [LM17] pour compiler PIC en un circuit d-DNNF [Dar01] nommé $dDNNF$. Cela nous permet de calculer à la fois le nombre de raisons suffisantes pour $\boldsymbol{x}$ étant donné T (il est donné par $\#(dDNNF)$, le nombre de modèles de $dDNNF$) et le pouvoir explicatif de chaque littéral $\ell \in t_{\boldsymbol{x}}$ (c'est le ratio du nombre de raisons suffisantes $\#(\{t \in E_s(\boldsymbol{x}, T) : \ell \in t\})$ pour $\boldsymbol{x}$ étant donné T qui contiennent ℓ – ce nombre coïncide avec le nombre de modèles $\#(dDNNF|\ell)$

14. Notons qu'en vertu de la construction, le pouvoir explicatif $Imp_s(\ell, \boldsymbol{x}, f)$ de ℓ pour $\boldsymbol{x}$ étant donné un classifieur booléen f est égale à 0 lorsque ℓ n'est pas un littéral de $t_{\boldsymbol{x}}$, car les raisons suffisantes pour $\boldsymbol{x}$ étant donné f sont des termes contenant uniquement des littéraux de $t_{\boldsymbol{x}}$.

de $dDNNF$ conditionné par ℓ – divisé par le nombre $\#(dDNNF)$ de raisons suffisantes pour $\boldsymbol{x}$ étant donné T).

La phase de compilation en d−DNNF est coûteuse en termes de temps de calcul dans le cas général, mais la dernière étape peut être accomplie en temps polynomial par rapport à la taille de $dDNNF$. En effet, le langage d−DNNF prend en charge en temps polynomial la requête de comptage des modèles et la transformation qu'est le conditionnement [DM02].

Algorithm 4 Calculer le pouvoir explicatif des littéraux

Input: un arbre de décision $T \in \mathtt{DT}_d$ et une instance $\boldsymbol{x} \in \{0,1\}^d$.
Output: le pouvoir explicatif Imp_s de chaque littéral de $t_{\boldsymbol{x}}$.

$C = \{c \cap t_{\boldsymbol{x}} : c \in \mathtt{CNF}(T)\}$
/ ⋆*calculer une formule* CNF *C tel que les impliquants premiers de C sont les raisons suffisantes pour $\boldsymbol{x}$ étant donné T* ⋆ /

$PIC = CNF\_PI(C)$
/ ⋆*traduire C dans une formule* CNF *PIC avec des modèles représentant les impliquants premiers de C [JMSSS14]* ⋆ /

$dDNNF = \mathtt{D4}(PIC)$
/ ⋆*compiler PIC dans un circuit* d−DNNF *$dDNNF$ [Dar01, LM17]* ⋆ /

for $\ell \in L$ **do**
  $Imp_s(\ell, \boldsymbol{x}, T) = \frac{\#(dDNNF|\ell)}{\#(dDNNF)}$
  / ⋆*calculer le pouvoir explicatif des littéraux à partir de $t_{\boldsymbol{x}}$* ⋆ /

return (Imp_s)

Exemple 39. *Illustrons l'algorithme 4 à l'aide de notre exemple 34 en nous concentrant sur l'instance $\boldsymbol{x} = (1,1,1,1)$. À partir de la représentation* CNF *de T donnée dans (33), nous obtenons :*

$$C = (x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (x_1 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4) \wedge (x_2 \vee x_4) \wedge (x_3 \vee x_4) \wedge x_4.$$

qui peut être simplifiée en la formule équivalente $(x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge x_4$ en utilisant la propagation unitaire. Ensuite, PIC est obtenu en ajoutant (conjonctivement) à C les implications suivantes (une formule est générée par littéral apparaissant dans C) : $x_1 \rightarrow (\overline{x}_2 \vee \overline{x}_3)$, $x_2 \rightarrow \overline{x}_1$, et $x_3 \rightarrow \overline{x}_1$. Chaque implication précise les conditions dans lesquelles le littéral correspondant ℓ (la condition de l'implication) participe à un impliquant premier de C, c'est-à-dire qu'il existe une clause de C qui est satisfaite par ℓ mais la clause n'est plus satisfaite si ℓ est supprimé. Dans le cas général, la partie conclusion de chaque implication est une formule DNF *(équivalente à la négation de toutes les clauses de C privées de ℓ), de sorte que l'implication elle-même ne peut pas être lue directement comme une formule* CNF *: de nouvelles variables doivent être introduites en utilisant la transformation de Tseitin pour transformer chaque implication en une formule* CNF *ayant les mêmes conséquences sur l'ensemble initial de variables que l'implication correspondante. La formule* CNF *résultante est PIC. Dans l'exemple, l'introduction de nouvelles variables est inutile car chaque implication peut être considérée comme une clause. Ainsi, on obtient la formule* CNF *$PIC = (x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge x_4 \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3) \wedge (\overline{x}_2 \vee \overline{x}_1) \wedge (\overline{x}_3 \vee \overline{x}_1)$. Enfin, PIC peut être compilé dans le circuit* d−DNNF *$dDNNF$ donné à la figure 27. Ce circuit $dDNNF$ a deux modèles sur $\{x_1, x_2, x_3, x_4\}$, à savoir $(0,1,1,1)$ et $(1,0,0,1)$. Chacun d'eux correspond (de manière biunivoque) à une raison suffisante pour $\boldsymbol{x}$ étant donné T, donnée par les littéraux de $t_{\boldsymbol{x}}$ qui y apparaissent. Ainsi, $(0,1,1,1)$ est associé à $x_2 \wedge x_3 \wedge x_4$, et $(1,0,0,1)$ est associé à $x_1 \wedge x_4$. Les*

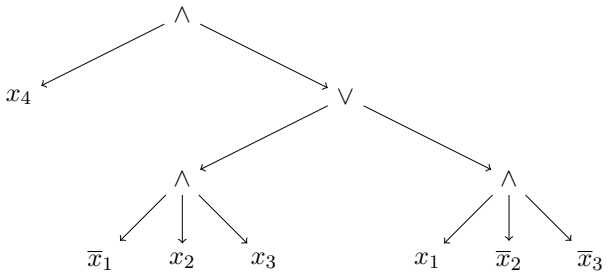


FIGURE 27 – Un circuit d−DNNF représentant les raisons suffisantes pour $\boldsymbol{x}$ étant T.

deux raisons suffisantes pour $\boldsymbol{x}$ étant donné T sont récupérées, comme prévu. À partir de $dDNNF$, nous pouvons facilement calculer le pouvoir explicatif des littéraux ℓ apparaissant dans $t_{\boldsymbol{x}}$ en calculant pour chacun d'eux le rapport $\frac{\#(dDNNF|\ell)}{\#(dDNNF)}$. Nous obtenons $Imp_s(x_1, \boldsymbol{x}, T) = \frac{1}{2}$, $Imp_s(x_2, \boldsymbol{x}, T) = \frac{1}{2}$, $Imp_s(x_3, \boldsymbol{x}, T) = \frac{1}{2}$, et $Imp_s(x_4, \boldsymbol{x}, T) = 1$.

Nous montrons à la section 7.3 que, malgré une complexité élevée dans le pire des cas (la taille de $dDNNF$ peut être exponentielle en $|T|$), cette approche basée sur la compilation des connaissances se révèle très efficace en pratique.

Assez clairement, une fois que $Imp_s(\ell, \boldsymbol{x}, T)$ a été calculé pour chaque $\ell \in t_{\boldsymbol{x}}$, on peut facilement générer des cartes de chaleur explicatives. La sous-figure la plus à droite de la figure 26 présente la carte de chaleur explicative calculée pour l'instance mnist correspondant à la sous-figure la plus à gauche. Cette instance comporte 12 littéraux nécessaires et 105 littéraux pertinents. Les pixels bleus (resp. rouges) correspondent aux littéraux positifs (resp. négatifs) dans l'instance, et l'intensité de la couleur vise à refléter le pouvoir explicatif du littéral correspondant. Cette carte de chaleur explicative offre une explication synthétique à la raison pour laquelle l'instance correspondante est reconnue comme un « 4 », et non comme un « 9 ». Elle suggère que l'instance est un « 4 » car les pixels « fermant la boucle » d'un « 9 » et ceux formant la « queue » d'un « 9 » sont désactivés.

Remarque 12 (complexité du calcul du pouvoir explicatif). *Pour le calcul du pouvoir explicatif, nous n'avons pas mis en avant de preuve de difficulté. Nous n'avons pas attaqué la question, mais nous supposons qu'il n'existe pas d'algorithme en temps polynomial pour cela. Ainsi, nous conjecturons que le problème du calcul du pouvoir explicatif est difficile, c'est-à-dire qu'il n'existe pas d'algorithme en temps polynomial pour réaliser son calcul.*

6.3 Calculer toutes les explications contrastives

Il a été démontré que les raisons suffisantes et les explications contrastives sont reliées par une dualité d'ensemble intersectant minimal (*minimal hitting set*) [INAMS20b]. Cette dualité peut être exploitée pour dériver l'un des deux ensembles d'explications à partir de l'autre en utilisant des algorithmes de calcul d'ensemble intersectant minimal [Rei87].

Cependant, dans le cas des arbres de décision, une approche plus directe et beaucoup plus efficace pour dériver l'ensemble de toutes les explications contrastives pour $\boldsymbol{x} \in \{0,1\}^d$ étant donné $T \in \mathtt{DT}_d$ peut être mise en place. En effet, contrairement à ce qui se passe pour les raisons suffisantes, l'ensemble de *toutes* les explications contrastives pour $\boldsymbol{x} \in \{0,1\}^d$ étant donné un arbre de décision $T \in \mathtt{DT}_d$ peut être calculé en temps polynomial à partir de $\boldsymbol{x}$ et de T. Notons que ce résultat a également été obtenu en parallèle et indépendamment de nous (voir [HIIMS21]). Il est important de noter que dans le contexte des arbres de décision, le nombre d'explications contrastives de taille minimale possibles pour

une instance donnée $\boldsymbol{x} \in \{0,1\}^d$ ne peut pas dépasser la taille de l'arbre de décision T lui-même, noté $|T|$. Cela découle de la structure même des arbres de décision, où chaque chemin de la racine à une feuille représente une trajectoire de décision spécifique pour une instance donnée. Étant donné que les explications contrastives sont des variations de cette trajectoire de décision, elles sont nécessairement limitées par les chemins disponibles dans l'arbre. Par conséquent, le nombre d'explications contrastives pour une instance ne peut pas excéder la taille de l'arbre de décision. Cette propriété souligne à quel point les explications contrastives sont intrinsèquement liées à la structure de l'arbre de décision utilisé pour la classification.

Rappel $\min(\{.\}, \subseteq)$**.** Pour une meilleure compréhension de la proposition 21 et de sa preuve, rappelons que la notation $\min(\{.\}, \subseteq)$ désigne l'ensemble des sous-ensembles de $\{.\}$ qui sont minimaux pour $\subseteq$. L'exemple 40 illustre cela.

Exemple 40. *Soit la* CNF $\varphi = (x_1 \vee x_2 \vee x_3) \wedge x_1 \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3})$*, où les clauses de la* CNF φ *sont vues comme des ensembles de littéraux* $\{\{x_1, x_2, x_3\}, \{x_1\}, \{\overline{x_1}, x_2\}, \{\overline{x_1}, x_2, \overline{x_3}\}\}$*. Nous remarquons que l'ensemble* $\{x_1\}$ *est inclus dans l'ensemble* $\{x_1, x_2, x_3\}$*, et que l'ensemble* $\{\overline{x_1}, x_2\}$ *est inclus dans l'ensemble* $\{\overline{x_1}, x_2, \overline{x_3}\}$*. Donc, nous avons :*

$$min(\{c : c \in \varphi\}, \subseteq) = \{\{x_1\}, \{\overline{x_1}, x_2\}\}$$

Proposition 21. *L'ensemble de toutes les explications contrastives pour* $\boldsymbol{x} \in \{0,1\}^d$ *étant donné un arbre de décision* $T \in \mathtt{DT}_d$ *peut être calculé en temps polynomial en* $d + |T|$ *comme*

$$min(\{c \cap t_{\boldsymbol{x}} : c \in \mathtt{CNF}(T)\}, \subseteq)$$

lorsque $T(\boldsymbol{x}) = 1$*, et comme*

$$min(\{c \cap t_{\boldsymbol{x}} : c \in \mathtt{CNF}(\neg T)\}, \subseteq)$$

lorsque $T(\boldsymbol{x}) = 0$*.*

Preuve 6. *Par définition, les raisons suffisantes* t *pour* $\boldsymbol{x}$ *étant donné* T *sont des impliquants premiers de* T *qui couvrent* $\boldsymbol{x}$*. Ainsi, elles sont précisément les impliquants premiers de l'ensemble de clauses* $\{c \cap t_{\boldsymbol{x}} : c \in \mathtt{CNF}(T)\}$ *où* $\mathtt{CNF}(T)$ *est une formule* CNF *équivalente à* T*. Donc la raison complète pour* $\boldsymbol{x}$ *étant donné* T *(définie comme la disjonction de toutes les raisons suffisantes pour* $\boldsymbol{x}$ *étant donné* T *[DH20]) est une fonction booléenne monotone, disons* f*.*

Les impliqués premiers d'une telle fonction monotone f *sont précisément les ensembles intersectants minimaux (minimal hitting sets) des impliquants premiers de* f*. Comme les explications contrastives pour* $\boldsymbol{x}$ *étant donné* T *sont aussi les ensembles intersectants minimaux des raisons suffisantes pour* $\boldsymbol{x}$ *étant donné* T *[INAMS20b], les explications contrastives pour* $\boldsymbol{x}$ *étant donné* f *sont les ensembles de littéraux correspondant aux impliquants premiers de* f*.*

Maintenant, puisque l'ensemble de clauses $\{c \cap t_{\boldsymbol{x}} : c \in \mathtt{CNF}(f)\}$ *est équivalent à* f *qui est une fonction monotone, ses impliqués premiers sont ses éléments minimaux pour l'inclusion* $\subseteq$*. Cela résulte de la correction des algorithmes basés sur la résolution pour générer des impliqués premiers (voir par exemple [Mar00]). En effet, il n'ya pas de résolution possible entre les clauses de* $\{c \cap t_{\boldsymbol{x}} : c \in \mathtt{CNF}(f)\}$ *car l'ensemble de clauses est monotone, donc il suffit de réaliser des tests de sous-sommation entre les clauses pour générer les impliqués premiers. Finalement,* $\{c \cap t_{\boldsymbol{x}} : c \in \mathtt{CNF}(T)\}$ *peut être calculé en temps polynomial en* $d + |T|$ *car* $\mathtt{CNF}(T)$ *peut être calculé en temps linéaire dans* $|T|$*. En utilisant un temps quadratique supplémentaire dans la taille de cet ensemble* $\{c \cap t_{\boldsymbol{x}} : c \in \mathtt{CNF}(T)\}$*, ses éléments minimaux pour l'inclusion* $\subseteq$ *peuvent être sélectionnés. L'ensemble obtenu est par construction l'ensemble de toutes les explications contrastives pour* $\boldsymbol{x}$ *étant donné* T*, et cet ensemble a été calculé en temps polynomial en* $d + |T|$*.*

Exemple 41. *Revenons à notre exemple 34, utilisons encore une fois la* CNF *représentant l'arbre* T *et l'instance* $\boldsymbol{x} = (1, 1, 1, 1)$. *Nous avons* $min(\{c \cap t_{\boldsymbol{x}} : c \in \texttt{CNF}(T)\}, \subseteq) = \{x_1 \vee x_2, x_1 \vee x_3, x_4\}$, *ce qui correspond aux explications contrastives* $x_1 \wedge x_2$, $x_1 \wedge x_3$, *et* x_4 *pour* $\boldsymbol{x}$ *étant donné* T *quand on considère les clauses et les termes comme des ensembles de littéraux. De même, nous avons*

$$\texttt{CNF}(\neg T) = \{x_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_4, \overline{x}_1 \vee x_2 \vee x_3 \vee \overline{x}_4, \overline{x}_1 \vee x_2 \vee \overline{x}_3 \vee \overline{x}_4, \overline{x}_1 \vee \overline{x}_2 \vee x_3 \vee \overline{x}_4, \overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_4\}.$$

Ainsi, avec $\boldsymbol{x}' = (0, 0, 0, 0)$, *nous avons* $min(\{c \cap t_{\boldsymbol{x}'} : c \in \texttt{CNF}(\neg T)\}, \subseteq) = \{\overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_4, \overline{x}_1 \vee \overline{x}_4\}$; *ce qui correspond aux explications contrastives* $\overline{x}_2 \wedge \overline{x}_3 \wedge \overline{x}_4$ *et* $\overline{x}_1 \wedge \overline{x}_4$ *pour* $\boldsymbol{x}'$ *étant donné* T.

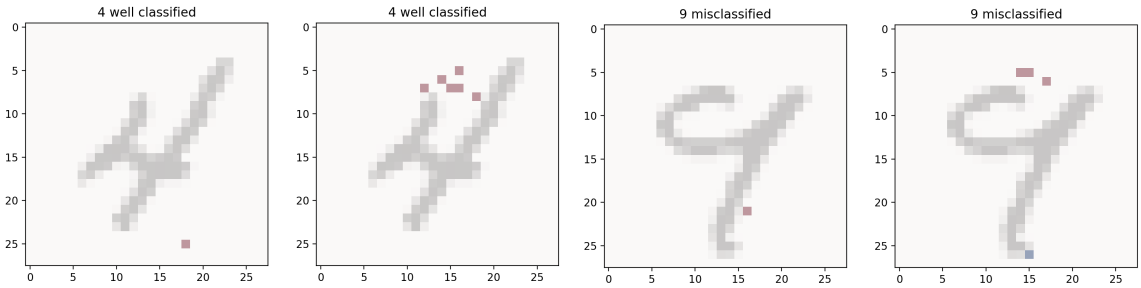


FIGURE 28 – Deux exemples provenant de `mnist49` : un « 4 » correctement identifié comme un « 4 » par l'arbre de décision, et un « 9 » mal classifié comme un « 4 » par l'arbre de décision. Pour chaque instance, deux explications contrastives sont présentées (les premières à gauche correspondent au « 4 » correctement classifié, et les dernières à droite correspondent au « 9 » mal classifié).

À titre d'illustration, la figure 28 présente deux instances issues de `mnist49` qui ont été prises en compte dans nos expériences : la première instance est un « 4 » correctement identifié comme un « 4 » par l'arbre de décision, et la deuxième instance est un « 9 » mal classifié comme un « 4 » par l'arbre de décision. On peut observer que le « 4 » ressemble vraiment à un « 4 », tandis que le « 9 » pourrait aussi être interprété comme un « 4 ». Le « 4 » correctement classifié possède 42 explications contrastives données par l'arbre de décision : 10 de taille 1, 11 de taille 2, 11 de taille 3, 5 de taille 4, 3 de taille 5 et 2 de taille 6. Le « 9 » mal classifié a 21 explications contrastives données par l'arbre de décision : 3 de taille 1, 5 de taille 2, 10 de taille 3 et 3 de taille 4. Pour ces deux instances, la taille des explications contrastives est plutôt réduite (avec une taille maximale de 6 pour la première instance), étant donné que 784 attributs sont utilisés pour décrire les instances (les images sont composées de 28×28 pixels). Pour chaque instance, deux explications contrastives sont présentées sur la figure. L'image la plus à gauche montre une explication contrastive de taille 1 pour l'instance « 4 » en question. Cette première explication n'est pas très convaincante : on peut difficilement voir pourquoi activer le pixel rouge changerait le « 4 » en un « 9 ». Cela montre que notre classifieur n'est pas aussi robuste qu'on pourrait le souhaiter. L'image suivante (de gauche à droite) présente une autre explication contrastive (de taille 6) pour la même instance « 4 ». Cette explication est intuitivement meilleure que la précédente (en gros, elle indique que fermer la boucle" en activant les six pixels rouges donnés par l'explication suffirait à changer la prédiction de « 4 » en « 9 »). De même, les deux dernières images montrent deux instances contrastives pour le « 9 » mal classifié. La première a une taille de 1 et la deuxième (correspondant à l'image la plus à droite) a une taille de 4. La deuxième explication semble légèrement meilleure que la première, qui n'est clairement pas satisfaisante.

Comme illustré par la figure 28, la proposition 21 a deux conséquences directes, mais remarquables : d'une part, le nombre d'explications contrastives pour n'importe quelle instance étant donné un arbre

TABLE 5 – Description des 20 jeux de données pour lesquels des résultats empiriques sont fournis.

Dataset	#I	#F	#C	source
recidivism	26020	11	2	Kaggle
adult	48842	14	2	UCI
bank marketing	45211	17	2	UCI
bank	41188	21	2	Kaggle
lending loan	9578	13	2	Kaggle
contraceptive	1473	9	3	UCI
compas	5278	14	2	OpenML
christine	5418	1637	2	OpenML
farm-ads	4143	54877	2	UCI
mnist49	13782	784	2	-
spambase	4601	57	2	UCI
mnist38	13966	784	2	-
madelon	4400	500	2	UCI
gisette	7000	5000	2	OpenML
gina	3153	970	2	OpenML
price phone	2000	20	4	Kaggle
letter	20000	16	2	UCI
titanic	623	5	2	Kaggle
yeast	2417	117	2	OpenML
soybean	683	35	2	UCI

de décision ne peut pas dépasser le nombre de branches dans l'arbre, d'autre part, la taille de toute explication contrastive pour n'importe quelle instance étant donné un arbre de décision est bornée supérieurement par la profondeur de l'arbre. Ces deux propriétés offertes par les arbres de décision sont précieuses du point de vue de l'intelligibilité. Cependant, la deuxième propriété peut également être considérée comme un témoignage de la *limitation intrinsèque de la robustesse des arbres de décision* en tant que modèle d'apprentissage automatique : *modifier quelques attributs (pas plus que la profondeur de l'arbre) dans l'instance d'entrée (quelle qu'elle soit) suffit pour changer la prédiction effectuée*. Pour une profondeur fixée, cela est vrai *quelle que soit la précision de l'arbre*. Ainsi, bien que la précision moyenne de l'arbre de décision utilisé pour classifier les deux instances de la figure 28 dépasse 95% sur l'ensemble de test (voir la section 7.3), activer un seul pixel suffit pour que le classifieur reconnaisse un « 9 » au lieu d'un « 4 » pour la première instance, et un « 4 » au lieu d'un « 9 » pour la deuxième instance. D'autres conséquences directes de la proposition 21 sont que le calcul des attributs nécessaires / pertinents et le calcul de l'importance explicative des attributs par rapport aux explications contrastives peuvent être réalisés en temps polynomial en $d + |T|$. En effet, puisqu'elles peuvent être énumérées efficacement, les explications contrastives peuvent également être facilement minimisées et comptées.

6.4 Expérimentations

Protocole expérimental. Nous avons considéré 90 jeux de données (aussi appelés *datasets* ou *benchmarks*) bien connus et disponibles depuis Kaggle (www.kaggle.com), OpenML (www.openml.org), et UCI (archive.ics.uci.edu/ml/). *mnist38* et *mnist49* sont deux sous-ensembles de mnist, restreints aux instances des chiffres 3 et 8 et respectivement 4 et 9).Certains ensembles de données

TABLE 6 – Résultats empiriques pour les 20 jeux de données.

	Decision Tree				| Sufficient |		| Min.-Sized |		#Nec. Features		#Rel. Features	
Dataset	%A	#N	#B	#D	med	max	med	max	med	max	med	max
recidivism	63.4	13828.8	147.6	27.8	14	22	13	22	6	19	60	98
adult	81.4	12934.0	2974.8	47.0	16	36	16	36	7	22	263	543
bank marketing	87.4	6656.4	1432.6	30.7	14	21	14	21	3	16	247	398
bank	89.0	5523.6	977.8	29.6	13	24	13	24	4	15	200	330
lending loan	73.5	2610.4	1131.4	33.3	16	31	16	31	8	25	226	442
contraceptive	50.4	1252.2	88.6	22.0	11	20	11	20	8	17	25	47
compas	66.0	1230.0	46.2	19.1	6	14	6	14	3	12	16	33
christine	63.4	853.2	426.0	36.6	12	47	12	47	8	41	92	202
farm-ads	86.8	544.8	264.6	85.5	20	99	20	99	16	92	73	192
mnist49	95.5	539.6	267.9	28.8	22	30	22	30	9	19	91	166
spambase	92.0	536.4	264.8	30.1	15	29	15	29	9	24	68	146
mnist38	96.0	506.6	251.4	27.2	19	28	19	28	8	20	93	157
madelon	75.2	357.8	178.2	16.7	10	20	10	20	7	18	38	103
gisette	93.3	347.8	173.2	36.0	27	39	27	39	19	34	64	113
gina	85.8	337.0	173.0	24.1	12	26	12	26	7	19	54	108
price phone	82.2	335.6	161.5	13.2	8	14	8	14	4	11	27	76
letter	99.3	317.0	95.9	15.8	6	15	6	15	1	12	49	81
titanic	75.9	274.0	116.3	17.1	7	17	7	17	4	14	22	58
yeast	97.3	68.8	33.9	18.7	15	20	15	20	7	16	26	38
soybean	96.9	46.2	19.3	11.6	2	9	2	9	1	8	8	19

	#Sufficient		#Contrastive		| Contrastive |		#Min.-Sized	
Dataset	med	max	med	max	med	max	med	max
recidivism	10387	9734080	54	145	3	16	2	144
adult	-	$\geq$ 157383572260730000000000	201	470	4	16	3	256
bank marketing	-	$\geq$ 74603752134843500000000	189	337	4	13	8	432
bank	-	$\geq$ 7433951979018500000	150	277	4	13	4	168
lending loan	459258918095775	943243242816203000000000000000	157	311	3	12	3	192
contraceptive	20,50	4272	21	52	2	11	2	48
compas	16	444	13	33	2	11	2	21
christine	63108	2167735434744	71	151	3	8	2	4096
farm-ads	1177,50	921895392	59	166	2	10	-	$\geq$ 10000
mnist49	7392384	715892613696000	61	106	2	12	-	$\geq$ 10000
spambase	15712	2535069312	50	107	2	11	4	384
mnist38	14849376	16922386736640	62	107	3	11	32	3072
madelon	106	3221020	30	72	2	9	2	32
gisette	3905	234593712	50	81	2	10	-	$\geq$ 10000
gina	4544	432967680	38	77	2	8	4	6144
price phone	109	363828	19	50	2	7	2	32
letter	9342	28391526	32	56	3	9	4	256
titanic	44	49920	16	38	2	9	2	96
yeast	128	24576	18	23	2	5	8	4608
soybean	3	60	5	15	2	6	1	20

concernent des problématiques de classification multi-classes. Dans de tels cas, nous avons utilisé la stratégie standard du « un contre tous » pour les gérer, toutes les classes sauf la classe cible sont considérées comme la classe complémentaire de la cible. Aucun pré-traitement n'a été effectué sur les attributs numériques, ces derniers ont été binarisés à la volée par l'algorithme d'apprentissage des arbres de décision utilisé. Pour chaque benchmark b, un processus de validation croisée à 10-blocs a été réalisé. Plus précisément, un ensemble de dix arbres de décision T_b a été calculé et évalué à partir des instances étiquetées de b, qui ont été réparties en 10 parties. Chaque partie a été utilisée comme ensemble de test et les 9 autres parties comme ensemble d'entraînement pour générer un arbre de décision. Chaque T_b correspond ainsi à l'ensemble de test choisi parmi l'ensemble complet des données de b. La performance de classification pour b a été mesurée comme la précision moyenne obtenue sur les 10 arbres de décision générés à partir de b. L'algorithme CART, et plus spécifiquement son implémentation fournie par la bibliothèque Scikit-Learn [PVG+11], a été utilisée pour apprendre les arbres de décision. Tous les hyper-paramètres de l'algorithme d'apprentissage ont été réglés sur leurs valeurs par défaut. Les arbres de décision ont été appris en utilisant l'indice de Gini, et sans aucune profondeur maximale ni autre changement manuel.

Pour chaque benchmark b, chaque arbre de décision T_b, et un sous-ensemble d'au plus 100 instances $\boldsymbol{x}$ prises au hasard dans l'ensemble de test suivant une distribution uniforme, nous avons calculé les raisons suffisantes pour $\boldsymbol{x}$ étant donné T_b (en utilisant l'algorithme glouton standard prenant comme entrée la raison directe $t_{\boldsymbol{x}}^{T_b}$) et les raisons suffisantes minimales pour $\boldsymbol{x}$ étant donné T_b en utilisant le codage

PARTIAL MAXSAT présenté dans la proposition 19. Cela nous a permis d'obtenir des statistiques (médiane, maximum) sur les tailles des raisons générées. En utilisant l'algorithme présenté dans la preuve de la proposition 20, nous avons également calculé le nombre d'attributs nécessaires et pertinents de chaque instance considérée $\boldsymbol{x}$ et avons de nouveau obtenu des statistiques à leur sujet. En exploitant le compteur de modèles `D4`, nous avons calculé le nombre de raisons suffisantes pour $\boldsymbol{x}$ étant donné T_b, ainsi que le pouvoir explicatif de chaque attribut de $\boldsymbol{x}$. En utilisant l'algorithme présenté dans la proposition 6.3, nous avons calculé le nombre d'explications contrastives pour $\boldsymbol{x}$ étant donné T_b, et obtenu des statistiques sur ces nombres ainsi que sur les tailles des explications contrastives. Enfin, en utilisant l'approche décrite à la section 6.3, nous avons énuméré toutes les raisons suffisantes minimales pour $\boldsymbol{x}$ étant donné T_b jusqu'à une limite définie à 10000, et avons encore obtenu des statistiques sur les nombres de raisons suffisantes minimales. Pour chaque calcul, nous avons mesuré les temps d'exécution correspondants, car cela est fondamental pour déterminer dans quelle mesure les algorithmes sont efficaces en pratique.

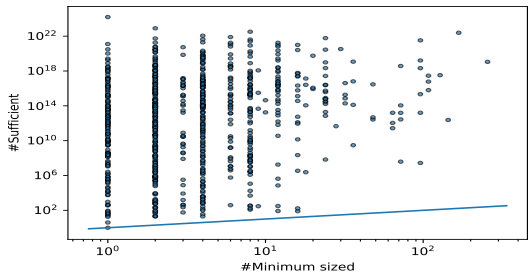
Toutes les expériences ont été effectuées sur un ordinateur équipé d'un processeur Intel(R) XEON E5-2637 CPU @ 3,5 GHz et de 128 Go de mémoire. `D4` [LM17] a été exécuté avec ses paramètres par défaut. Un temps limite (TO) de $100(s)$ secondes par instance a été utilisé pour `D4`. Pour calculer les raisons minimales, nous avons utilisé la bibliothèque Pysat (`https://pysathq.github.io/docs/html/index.html`), qui fournit la mise en œuvre du solveur RC2 (PARTIAL MAXSAT). Ce solveur a été exécuté en utilisant les paramètres correspondant au mode « Glucose ».

Résultats. Nous présentons un extrait de nos résultats, en nous concentrant sur 20 jeux de données parmi les 90 disponibles. Les jeux de données sélectionnés font partie de ceux contenant un grand nombre d'instances et / ou un grand nombre d'attributs (datasets de grande dimension). Ils sont décrits dans le tableau 5. La colonne la plus à gauche de ce tableau indique le nom du dataset. Les colonnes $\#I$, $\#F$, $\#C$, et source donnent respectivement le nombre d'instances dans le dataset, le nombre d'attributs utilisées pour décrire les instances, le nombre de classes, et l'origine du benchmark.

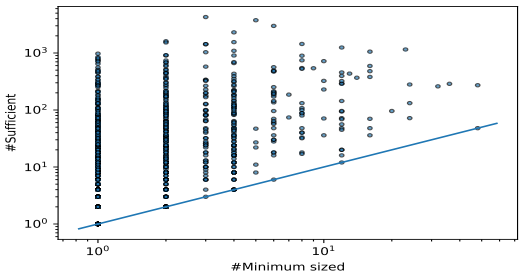
Le tableau 6 (haut et bas) présente les résultats obtenus pour les 20 datasets du tableau 5. La colonne la plus à gauche du tableau 6 indique le nom de dataset b. Les colonnes $\%A$, $\#N$, $\#B$, et $\#D$ donnent respectivement la précision moyenne sur les dix arbres de décision, le nombre moyen de nœuds dans ces arbres, le nombre moyen d'attributs binaires, et la profondeur moyenne de leurs branches. Les colonnes suivantes donnent des statistiques (médiane, maximum) sur, respectivement, la taille de la première raison suffisante (| Sufficient |) et de la première raison suffisante minimale (| Min.-Sized |) qui ont été calculées pour l'instance considérée. Ensuite, on trouve les nombres d'attributs nécessaires (#Nec. Features) et d'attributs pertinents (#Rel. Features) qui apparaissent dans l'ensemble des raisons suffisantes de l'instance. Le tableau 6 (bas) donne des statistiques (médiane, maximum) sur, respectivement, les nombres de raisons suffisantes (#Sufficient) qui ont été calculées, les nombres d'explications contrastives (#Contrastive) et leurs tailles (| Contrastive |), et enfin les nombres de raisons suffisantes minimales (#Min-Sized).

La figure 29 présente des graphiques en nuages de points pour comparer les nombres de raisons suffisantes avec les nombres de raisons suffisantes minimales pour des instances de 4 datasets, à savoir `adult`, `contraceptive`, `lending loan` et `spambase`. Chaque point correspond à une instance (sur 1000) de l'ensemble de données test correspondant pour laquelle aucun temps limite (TO) n'a été atteint. La coordonnée x d'une instance représente le nombre de raisons suffisantes minimales de l'instance, et sa coordonnée y représente le nombre de raisons suffisantes. La ligne bleue dans chaque graphique en nuages de points rassemble les points (x, y) tels que $y = x$ (notons que différentes échelles sont utilisées sur les deux axes pour rendre les graphiques plus lisibles).

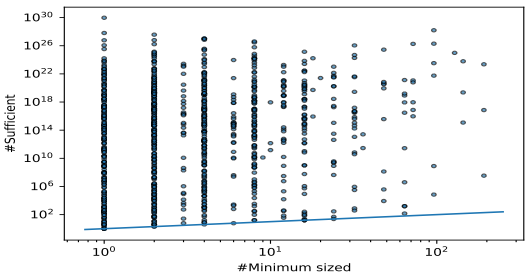
La figure 30 présente des boîtes à moustaches (*boxplot*) pour comparer les tailles des raisons suffi-



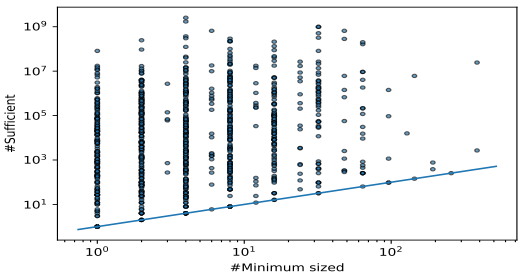
(a) `adult`



(b) `contraceptive`



(c) `lending loan`



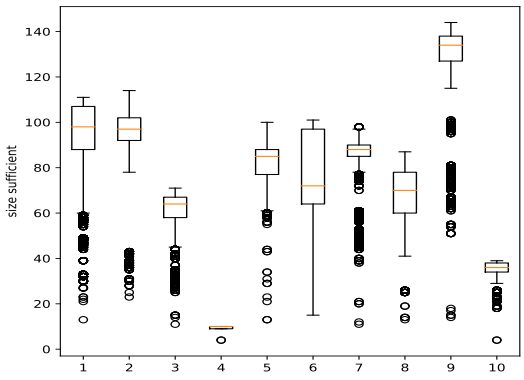
(d) `spambase`

FIGURE 29 – Comparaison du nombre de raisons suffisantes avec le nombre de raisons suffisantes minimales pour 4 datasets.

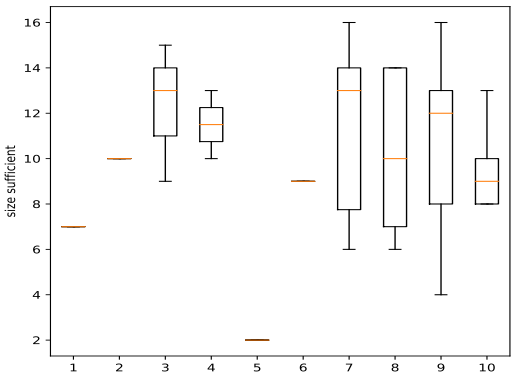
santes avec les tailles des raisons suffisantes minimales pour des instances des 4 datasets (les mêmes que celles considérées à la figure 29). Pour chaque dataset, dix instances ont été sélectionnées de manière aléatoire. Pour chacune d'entre elles, une raison suffisante minimale a été calculée, ainsi que des raisons suffisantes jusqu'à un maximum de 10000. La distribution des tailles de ces raisons est représentée dans un boxplot (la valeur minimale correspond à la taille d'une raison suffisante minimale). On peut observer des différences significatives entre les graphiques obtenus : pour certaines instances (par exemple, l'instance 9 de `adult`), la distribution obtenue est étalée (les raisons suffisantes minimales sont plus de sept fois plus petites que certaines raisons suffisantes), pour d'autres instances (par exemple, l'instance 4 de `adult`), la distribution est relativement étroite.

La table 6 montre également que, empiriquement, le nombre d'explications contrastives pour une instance est généralement bien plus petit que son nombre de raisons suffisantes. Elle montre également que les tailles des explications contrastives sont en général très petites. Cela est en accord avec le résultat énoncé à la proposition 21.

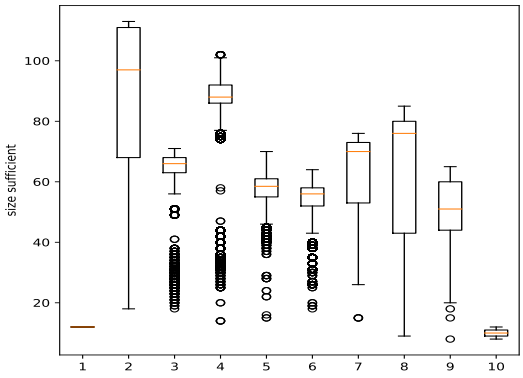
En ce qui concerne les temps de calcul, tous les algorithmes décrits dans les sections précédentes se sont révélés efficaces en pratique. Ce n'est pas surprenant pour les algorithmes ayant une complexité dans le pire des cas en temps polynomial (l'algorithme glouton pour calculer une raison suffisante, celui pour dériver le pouvoir explicatif des attributs et les attributs nécessaires et pertinents, et celui pour calculer toutes les explications contrastives). C'était moins évident à première vue pour les algorithmes utilisés pour compter le nombre de raisons suffisantes et pour calculer le pouvoir explicatif des attributs. Cependant, toutes les calculs qui ont été exécutés se sont finis en temps voulu, sauf pour les instances de



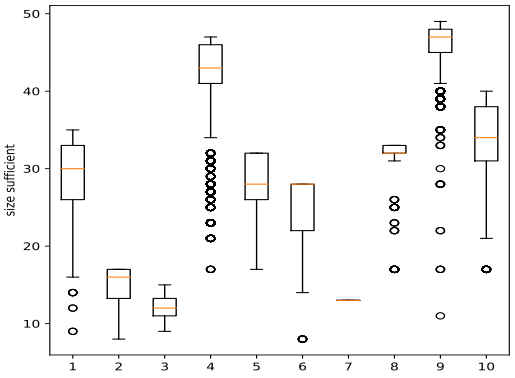
(a) `adult`



(b) `contraceptive`



(c) `lending loan`



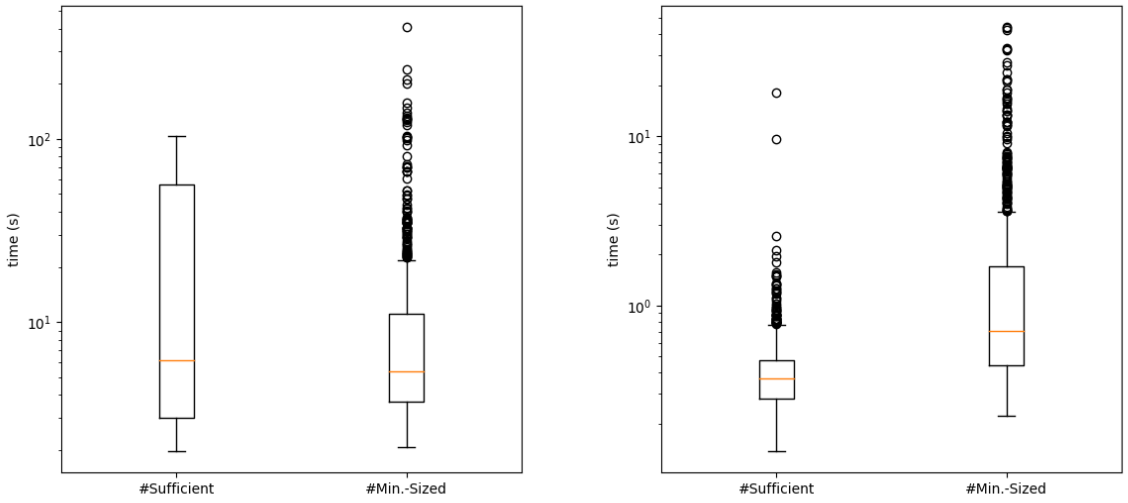
(d) `spambase`

FIGURE 30 – Comparaison des tailles des raisons suffisantes avec les tailles des raisons suffisantes minimales pour les instances de 4 datasets.

3 datasets sur 90, à savoir `adult`, `bank_marketing` et `bank`. Pour ces datasets, la limite de temps (TO) de $100s$ a été atteinte respectivement pour 203, 150 et 336 instances sur 1000 (dans ce cas, le nombre médian de raisons suffisantes n'a pas été calculé). Notamment, pour les 90 datasets sauf ces 3, le temps médian requis pour compter le nombre de raisons suffisantes et calculer le pouvoir explicatif des attributs n'a jamais dépassé $1s$.

Calculer une raison suffisante minimale, et plus généralement toutes ces raisons, semblait également être un défi en raison à la fois de la complexité intrinsèque du calcul d'une raison suffisante minimale et de leur nombre. Néanmoins, notre algorithme d'énumération a réussi à dériver *toutes les raisons suffisantes minimales* pour chaque ensemble de données, à l'exception de 3 sur 90, à savoir `farm-ads`, `mnist49` et `gisette`. Pour ces datasets, la limite de 10000 raisons a été atteinte respectivement pour 5, 16 et 3 instances sur 1000. Le temps médian nécessaire pour dériver toutes les raisons suffisantes minimales pour les instances pour lesquelles le calcul a réussi n'a dépassé $1s$ que pour 2 datasets (`adult`

et bank_marketing).



(a) adult (b) lending loan

FIGURE 31 – Distributions des temps de calcul nécessaires pour compter les raisons suffisantes et les raisons suffisantes minimales pour 1000 instances provenant de deux datasets.

La figure 31 présente des boîtes à moustaches (boxplots) illustrant la distributions des temps de calcul nécessaires pour compter les raisons suffisantes et les raisons suffisantes minimales pour 1000 instances provenant des deux datasets, adult et lending loan. L'approche présentée basée sur le compteur de modèles D4 a été utilisée pour compter le nombre de raisons suffisantes, et la méthode d'énumération avec des clauses bloquantes décrite précédemment a été exploitée pour compter les raisons suffisantes minimales. D4 a réussi à compter en un temps raisonnable $(100s)$ les raisons suffisantes pour 797 instances sur 1000. Lorsqu'il a échoué, le temps de calcul correspondant a été fixé à $100s$ pour établir les statistiques. La méthode d'énumération a réussi à calculer toutes les raisons suffisantes minimales pour chaque instance. La figure 31 illustre clairement que le temps requis pour compter les raisons suffisantes ou les raisons suffisantes minimales reste généralement raisonnable, malgré le nombre potentiellement élevé de raisons.

Nos expériences ont également mis en évidence que l'algorithme glouton pour dériver une raison suffisante calcule en pratique une raison suffisante minimale dans de nombreux cas. Cela explique la divergence entre les résultats rapportés dans le tableau 6 et ceux présentés à la figure 30. En effet, la taille d'une seule raison suffisante par instance (celle fournie par l'algorithme glouton) a été prise en compte pour calculer les résultats rapportés dans le tableau 6, tandis que toutes ses raisons suffisantes (jusqu'à un maximum de 10000) ont été prises en compte pour obtenir les valeurs utilisées pour dessiner la figure 30. Lorsque l'on considère l'ensemble complet des raisons, une différence considérable entre le nombre de raisons suffisantes et le nombre de raisons suffisantes minimales peut être observée (voir tableau 6 et figure 29). Se concentrer sur les raisons de taille minimale permet souvent de réduire considérablement le nombre de raisons, parfois de plusieurs ordres de grandeur. Pour certaines instances, le nombre de raisons suffisantes minimales est suffisamment petit pour qu'il soit envisageable de les fournir toutes à

l'utilisateur pour un examen plus approfondi. Cela est rarement le cas pour les raisons suffisantes.

Nos expériences ont également montré que le nombre d'attributs pertinents pour une instance est généralement beaucoup plus petit que le nombre d'attributs binaires, et que le nombre de d'attributs nécessaires est également nettement inférieur au nombre d'attributs pertinents. L'écart entre les deux explique le nombre potentiellement énorme de raisons suffisantes.

Enfin, tout comme les raisons suffisantes minimales, le nombre d'explications contrastives ne semble pas très élevé dans de nombreux cas. Cela concorde avec la limite théorique - le nombre de branches dans l'arbre ($|T|$) - soulignée à la section 6.3.

6.5 Conclusion

À la lumière de nos résultats, il s'avère que le pouvoir explicatif des arbres de décision va bien au-delà de leur capacité à générer efficacement des raisons directes. À partir d'un arbre de décision, le pouvoir explicatif des attributs et les raisons de taille minimale pour une instance peuvent être calculées efficacement en pratique la plupart du temps.

Pour être plus précis, d'après nos résultats, répondre à la question « Pourquoi pas ? » pour les arbres de décision semble plus facile que de répondre à la question « Pourquoi ? » : calculer l'ensemble complet des raisons suffisantes pour l'instance en question est généralement hors de portée, tandis que le calcul de son ensemble complet d'explications contrastives est faisable. En particulier, la faible robustesse des arbres de décision, c'est-à-dire le fait que la décision prise à propos d'une instance puisse être remise en question lorsqu'un petit nombre d'attributs de l'instance sont modifiées, peut s'expliquer par le fait que la taille de toute explication contrastive pour n'importe quelle instance donnée étant un arbre de décision est bornée supérieurement par la profondeur de l'arbre (et cette taille ne dépend pas directement de la précision de l'arbre).

Nos résultats montrent néanmoins que l'ensemble complet des raisons suffisantes pour une instance donnée par un arbre de décision peut être synthétisé efficacement la plupart du temps. Calculer les attributs nécessaires et pertinents par rapport aux raisons suffisantes est traitable. De plus, si la question de la traitabilité du calcul du pouvoir explicatif reste ouverte du point de vue théorique, nous avons mis en avant un algorithme qui permet souvent de réaliser ce calcul en pratique en un temps raisonnable.

Ainsi, le langage des arbres de décision apparaît non seulement comme attractif à des fins d'apprentissage lorsqu'il s'agit d'instances qui ne sont pas trop bruitées, mais aussi comme une bonne cible lorsque l'on a besoin de raisonner sur les différentes formes d'explication (abductives et contrastives) associées aux prédictions effectuées. Cela coïncide avec (et complète) les résultats rapportés au chapitre 5, montrant que de nombreuses tâches d'explication et de vérification sont faisables en temps polynomial pour les arbres de décision.

Dans ce chapitre, l'accent a été mis sur les explications de taille minimale. La signification de telles explications découle du rasoir d'Ockham : toutes choses étant égales par ailleurs, il est logique de préférer une explication plus courte à une plus longue, car une explication plus courte peut être considérée comme plus intelligible qu'une plus longue. Cependant, la taille n'est qu'un des critères à prendre en compte, et bien qu'il n'y ait pas de consensus sur ce qu'est une bonne explication [DK17, Lip18], il est clair que de nombreux autres critères qui ne sont pas intrinsèques aux explications mais qui dépendent fortement de l'utilisateur doivent également être pris en compte. En particulier, les préférences de l'utilisateur, lorsqu'elles sont disponibles, peuvent être utilisées pour sélectionner des explications de meilleure qualité. Dans un prochain chapitre, nous montrons que l'exploitation des préférences de l'utilisateur peut réduire considérablement le nombre d'explications abductives.

7

Explications abductives préférées

Sommaire

7.1 Explications abductives préférées 126
7.1.1 Modèles de préférences 126
7.1.2 Explications abductives préférées pour les arbres de décision 128
7.2 L'impact des fonctions de poids sur les raisons abductives préférées 129
7.2.1 Fonctions de poids 130
7.2.2 Transformation monotone 131
7.3 Résultats expérimentaux 131
7.3.1 Protocole expérimental 131
7.3.2 Résultats expérimentaux 133
7.4 Conclusion 135

Les explications abductives constituent un objet d'étude important du domaine de l'IA explicable dans la mesure où elles indiquent pourquoi les instances correspondantes sont classées comme elles le sont. Cependant, une instance peut avoir un nombre exponentiel d'explications abductives de taille minimale, même pour les familles de classifieurs « intelligibles », tels que les arbres de décision. Comme les explications abductives d'une instance peuvent de plus totalement différer l'une de l'autre, fournir à l'utilisateur la première explication calculée n'est pas très satisfaisant. En effet, de meilleures explications pour cet utilisateur peuvent exister. Pour pallier ce problème, nous proposons de prendre en compte des préférences utilisateur dans le calcul d'explications abductives et de ne calculer que des *explications préférées*. Dans ce chapitre, plusieurs modèles de préférence sont mis en évidence et discutés. Pour chaque modèle, nous présentons et évaluons un algorithme de calcul des **raisons suffisantes préférées** pour les arbres de décisions. Nous montrons qu'en pratique les raisons suffisantes préférées d'une instance peuvent être beaucoup moins nombreuses que ses raisons suffisantes.

Une partie de ces travaux a donné lieu à la publication [ABB+22b].

Introduction. Le travail présenté dans ce chapitre repose sur deux hypothèses de recherche. Premièrement, toutes les explications d'une instance ne sont pas égales, certaines sont meilleures que d'autres. Deuxièmement, la qualité d'une explication n'est en général pas intrinsèque à l'explication, mais elle dépend de l'utilisateur à qui on la fournit. Sous ces hypothèses, notre approche consiste à exploiter *des préférences utilisateur*, pour dériver seulement des *explications préférées* des instances. Se focaliser sur des explications préférées présente deux avantages : d'une part, les explications indiquées sont

meilleures (elles collent aux préférences de l'utilisateur) et d'autre part, leur nombre peut être drastiquement plus petit que le nombre total d'explications (on peut donc parfois les donner toutes). Dans notre étude, plusieurs modèles de préférence sont proposés. Nous partons d'un modèle simple conduisant à des préférences dichotomiques sur les explications où les explications acceptables sont seulement celles construites sur un ensemble particulier d'attributs, nous présentons également un modèle où les explications acceptables sont celles satisfaisant une contrainte prédéfinie. L'étape suivante consiste à considérer des préférences plus élaborées, qui ne sont pas dichotomiques par essence mais sont plus graduelles. Un modèle de relation de préférence ordinale est obtenu en stratifiant l'ensemble des attributs. Enfin, un dernier modèle est présenté, basé sur une fonction d'utilité / coût linéaire sur les attributs. Ici, chaque attribut a un coût, et les coûts sont agrégés de manière additive. Cela conduit à une relation de préférence cardinale sur les explications.

7.1 Explications abductives préférées

7.1.1 Modèles de préférences

Définir *in abstracto* ce qu'est une explication « préférée » ou, au minimum, « suffisamment bonne » est difficile en général. Il n'y a pas de consensus à ce sujet et une dépendance claire à l'utilisateur est à prendre en compte, voir par exemple [DK17, Lip18].

Afin de se concentrer uniquement sur les explications « préférées » ou du moins sur celles qui sont « suffisamment bonnes », un modèle formel de préférences doit être défini et utilisé. Un tel modèle peut être plus ou moins sophistiqué et prendre différentes formes. Nous présentons plusieurs modèles ci-dessous et définissons des notions de raisons préférées basées sur eux. Ces raisons préférées peuvent être envisagées par rapport à l'ensemble complet des explications abductives, ou à des sous-ensembles de celui-ci, en particulier ceux contenant uniquement des raisons suffisantes. Bien que les notions de raisons préférées aient du sens pour n'importe quel classifieur booléen, nos résultats portent principalement sur les arbres de décisions.

Préférences dichotomiques sur les explications. Nous présentons d'abord deux modèles où les préférences de l'utilisateur sont *dichotomiques*, c'est-à-dire que les explications peuvent être partitionnées en deux ensembles : l'une contenant des raisons **« suffisamment bonnes »** et l'autre contenant des raisons jugées **« pas assez bonnes »**.

Se concentrer sur des attributs spécifiques. Un premier modèle conduisant à une préférence dichotomique consiste en la donnée d'un sous-ensemble $S \subseteq X_d$. Ce modèle est suffisant pour gérer des situations où l'utilisateur veut écarter les explications qui font référence à des *concepts non compréhensibles* (modélisés comme des attributs en dehors de S). De telles explications à rejeter peuvent faire référence à des attributs correspondant à des notions trop techniques, qui ne sont pas comprises par l'utilisateur (par exemple, un terme médical pour un patient qui n'est pas médecin), ou encore parce qu'ils ne sont pas documentés ou sont assez vagues par essence.

Définition 62. *Soient $f \in \mathcal{F}_d$, $S \subseteq X_d$ et $\boldsymbol{x} \in \{0,1\}^d$. Une* explication abductive basée sur S *pour $\boldsymbol{x}$ étant donnée f est une explication abductive t pour $\boldsymbol{x}$ étant donnée f telle que $Var(t) \subseteq S$.*

Une raison suffisante basée sur S étant donné f et une explication abductive basée sur S étant donné f qui est minimale pour l'inclusion ensembliste.

S'assurer que seules les explications basées sur les attributs de S sont générées est également utile pour atteindre d'autres objectifs. Ainsi, la présence de certains *éléments protégés* doit être évitée dans les

explications chaque fois que cela est possible, car l'impossibilité de laisser ces attributs de côté reflète précisément le fait que la décision prise était biaisée [DH20]. Par exemple, considérons dans une procédure d'admission à l'université, un candidat pour lequel la décision prise par le classeur est positive : si toute explication abductive de cette décision mentionne le fait que le candidat vient d'une **ville natale riche (élément protégé)**, la décision est biaisée. Par conséquent, « venant d'une ville natale riche » ne devrait pas appartenir à S. Au-delà des problèmes de compréhensibilité ou de biais, la présence d'attributs *non actionnables* doit être évitée dans les explications. Ne pas être actionnable signifie simplement que l'utilisateur ne peut pas (ou peut difficilement) changer la valeur de cet attribut dans l'instance pour laquelle une explication est recherchée. Il est facile de décider s'il existe une raison suffisante basée sur un ensemble prédéfini S d'attributs dans le cas des arbres de décisions.

Proposition 22. *Soient $T \in \mathtt{DT}_d$ et $\boldsymbol{x} \in \{0,1\}^d$. Pour un ensemble $S \subseteq X_d$, décider si une raison suffisante basée sur S pour $\boldsymbol{x}$ étant donnée T existe, et dériver une telle raison lorsque c'est le cas, peut être réalisé en temps $\mathcal{O}(d|T|)$ en utilisant un algorithme glouton.*

Étant donné qu'un arbre de décision est une forêt aléatoire spécifique avec un seul arbre, cette proposition est dérivée directement de la proposition 3 de [ABB+22b]. La proposition 22 montre qu'il est effectivement facile de dériver une raison suffisante basée sur un ensemble prédéfini S d'attributs dans le cas des arbres de décision.

Ne retenir que des explications voulues. Un autre modèle conduisant à une préférence dichotomique prend la forme d'une formule C (une contrainte sur X_d) telle que chaque explication doit satisfaire C pour être acceptée par l'utilisateur. Par exemple, une telle contrainte C peut refléter une norme qui doit être suivie.

Définition 63. *Soit $T \in \mathtt{DT}_d$, C une formule sur X_d, et $\boldsymbol{x} \in \{0,1\}^d$. Une* explication raison suffisante compatible avec C *pour $\boldsymbol{x}$ étant donné T est une raison suffisante t pour $\boldsymbol{x}$ tel que $t \models C$.*

Préférences graduelles sur les explications. Les deux modèles précédents induisent des préférences dichotomiques sur les explications. Si une telle séparation en deux classes est commode dans certains cas, on s'attend à *plus de gradualité* dans d'autres cas, afin (par exemple) d'éviter la présence de certains attributs dans les explications sans l'interdire pour autant.

Exploiter une stratification des attributs. Voici une approche pour définir des relations de préférence qui ne sont généralement pas dichotomiques. Considérons un pré-ordre total $\leq$ sur X_d, tel que $x_i \leq x_j$ signifie que l'attribut x_i est considéré comme au moins aussi important que x_j. $\leq$ peut être représenté par une *stratification* de X_d, c'est-à-dire, une partition ordonnée $S_1, \ldots, S_p$ des attributs de X_d telle que les x_i et x_j appartenant au même ensemble S_u de la partition vérifient $x_i \leq x_j$ et $x_j \leq x_i$, et quand S_u précède S_v dans la partition (i.e., $u < v$), pour chaque élément x_i de S_u et chaque élément x_j de S_v, on a $x_i < x_j$.

A partir de la stratification $S_1, \ldots, S_p$ de X_d induite par $\leq$, on peut définir une relation de préférence $\sqsubset$ sur les termes sur X_d en indiquant que $t \sqsubset t'$ si et seulement si [15] $\exists i \in \{1, \ldots, p\} \forall j \in \{1, \ldots, i-1\}$, $t[S_j] = t'[S_j]$ et $t[S_i] \subset t'[S_i]$. On peut vérifier que $\sqsubset$ est un ordre partiel strict (c'est-à-dire une relation irréflexive et transitive). Étant donné un ensemble de termes, les éléments minimaux de cet ensemble pour $\sqsubset$ correspondent intuitivement à ceux qui contiennent le moins possible (pour l'inclusion d'ensembles) de littéraux les moins préférés. Sur cette base, nous sommes maintenant prêts à définir une notion d'explication abductive préférée pour l'inclusion :

15. La construction rappelle celle utilisée pour caractériser les sous-théories préférées dans [Bre89].

Définition 64. *Soient $f \in \mathcal{F}_d$, $\leq$ un pré-ordre total sur X_d, et $\boldsymbol{x} \in \{0,1\}^d$.* Une explication abductive *t pour $\boldsymbol{x}$ étant donné f et $\leq$ préférée pour l'inclusion s'il n'existe aucune explication abductive t' pour $\boldsymbol{x}$ étant donnée f qui satisfait $t' \sqsubset t$.*

Dans le cas des arbres de décision, l'algorithme glouton utilisé dans [ABB+22d] pour générer des raisons suffisantes peut être transformé de telle manière qu'il génère des raisons suffisantes préférées étant donné $\leq$. En particulier :

Proposition 23. *Soient $T \in \mathtt{DT}_d$, $\leq$ un pré-ordre total sur X_d, et $\boldsymbol{x} \in \{0,1\}^d$. Calculer une raison suffisante préférée pour l'inclusion pour $\boldsymbol{x}$ étant donné T et $\leq$ peut être réalisé en temps $\mathcal{O}(|G| + d|T|)$ où G est le graphe $(X_d, \leq)$.*

Étant donné qu'un arbre de décision est une forêt aléatoire spécifique avec un seul arbre, la proposition 23 est dérivée directement de la proposition 4 de [ABB+22b].

Exploiter une fonction d'utilité sur les attributs. Une autre approche pour modéliser une relation de préférence sur un domaine combinatoire consiste à tirer parti d'une *fonction d'utilité* (ou d'une fonction de coût). Dans notre cadre, on associe une valeur de disutilité (un poids représentant un coût) à chaque attribut : plus le poids est élevé, moins l'attribut est intéressant. Cette fois, la relation de préférence résultante est un pré-ordre total sur les explications, les meilleures explications étant celles de coût minimal.

Définition 65. *Soit $f \in \mathcal{F}_d$. Soit $w : X_d \to \mathbb{N}^*$ un vecteur de poids, où un poids est associé à chaque attribut. Une* explication abductive de coût minimal *pour $\boldsymbol{x}$ étant donné f et w est une explication abductive t pour $\boldsymbol{x}$ et f minimisant $\Sigma_{x \in Var(t)} w(x)$.*

Contrairement à ce qui a été fait précédemment, on ne peut pas profiter d'une variante en temps polynomial d'un algorithme glouton pour dériver des raisons suffisantes de coût minimal. En effet, dans le cas général, le calcul d'une raison suffisante de coût minimal étant donné un arbre de décision est NP-difficile. Ceci découle du fait que le calcul d'une raison suffisante minimale t pour une instance étant donné un arbre de décision est une raison suffisante de coût minimal t pour une instance étant donné un arbre de décision et un vecteur de poids w_1 tel que pour chaque $i \in [d]$, $w_1(x_i) = 1$, et que la dérivation d'une raison suffisante minimale t pour une instance étant donné un arbre de décision est NP-difficile [ABB+22d].

7.1.2 Explications abductives préférées pour les arbres de décision

Dans cette section, nous nous concentrons sur les modèles de préférences graduelles et le calcul de raisons suffisantes préférées correspondantes pour les arbres de décision.

Raisons suffisantes de coût minimal. Nous proposons d'abord une méthode pour calculer et énumérer les raisons suffisantes de coût minimal pour une instance $\boldsymbol{x}$ étant donné un arbre de décision.

Définition 66 (raison suffisante de coût minimal). *Soit $T \in \mathtt{DT}_d$. Soit $w : X_d \to \mathbb{N}^*$ un vecteur de poids associé à chaque attribut. Une* raison suffisante de coût minimal *pour $\boldsymbol{x}$ étant donné T et w est un terme $t \subseteq t_{\boldsymbol{x}}$ minimisant $\Sigma_{x \in Var(t)} w(x)$.*

La proposition suivante [ABB+22b] généralise la proposition 19 (on retrouve la proposition 19 quand le vecteur de poids utilisé est w_1).

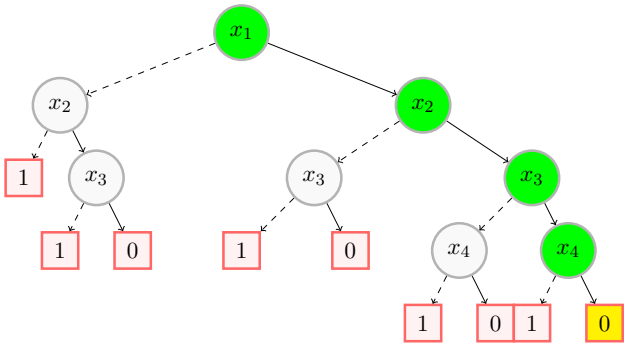


FIGURE 32 – Un arbre de décision pour classer les *prêts bancaires* en utilisant les attributs suivants : x_1 : « n'a pas de CDI », x_2 : « a plus de 50 ans », x_3 : « revenus annuels inférieurs à 35K » et x_4 : « n'a pas remboursé un précédent prêt ».

Proposition 24. *Soit $T \in \mathtt{DT}_d$ un arbre de décision et une instance $\boldsymbol{a} \in \{0,1\}^d$ tel que $T(\boldsymbol{a}) = 1$. Soit $w : X_d \to \mathbb{N}^*$ une application. Une raison suffisante de coût minimal pour $\boldsymbol{a}$ étant donné T et w est donnée par $t_{\boldsymbol{a}} \cap t_{x^*}$, où x^* est une solution de l'instance $(C_{\text{soft}}, C_{\text{hard}})$ du problème* WEIGHTED PARTIAL MAXSAT *tel que :*

$$C_{\text{soft}} = \{(\overline{x}_i, w(x_i)) : a_i = 1, i \in [d]\} \cup \{(x_i, w(x_i)) : a_i = 0, i \in [d]\}$$
$$C_{\text{hard}} = \{(c \cap t_{\boldsymbol{a}}, \infty) : c \in \mathtt{CNF}(T)\}$$

où $\mathtt{CNF}(T)$ est un encodage $\mathtt{CNF}$ de l'arbre de décision T. Comme déjà expliqué, si $\boldsymbol{a}$ vérifie $T(\boldsymbol{a}) = 0$, il suffit de considérer $\neg T$ à la place de T.

Nous tenons à souligner que les raisons suffisantes de coût minimal peuvent être énumérées avec la méme méthode que celle employée pour énumérer les raisons suffisantes minimales (voir chapitre 6). Cela peut être accompli en ajoutant une clause de blocage qui contrôle la somme des poids associé à chaque attribut de la première raison suffisante de coût minimal calculée, plutôt que de s'appuyer sur la contrainte liée à la taille.

Exemple 42. *Prenons l'exemple d'un banquier qui utilise l'arbre de décision de la figure 32 pour décider d'accorder ou non un prêt à un client. Le banquier souhaite comprendre pourquoi une instance particulière, $\boldsymbol{x} = (1,1,1,1)$, a été classée comme étant une instance négative ($T(\boldsymbol{x}) = 0$). Dans ce cas, il existe plusieurs raisons suffisantes pour expliquer cette classification. Ces raisons sont toutes les combinaisons d'attributs qui, si elles sont vraies, entraînent une classification négative. Dans le cas de $\boldsymbol{x} = (1,1,1,1)$, les raisons suffisantes sont : $x_1 \wedge x_2 \wedge x_4$, $x_1 \wedge x_3 \wedge x_4$ et $x_2 \wedge x_3 \wedge x_4$. Cependant, le banquier préfère une explication sans l'attribut x_2, car ce dernier est un attribut non actionnable, c'est-à-dire que le client ne peut pas le changer. Dans ce cas, nous pouvons utiliser une fonction de poids pour chaque attribut afin de trouver la meilleure explication. Dans cet exemple, nous utilisons la fonction de poids $w = (1,4,1,1)$, qui attribue des poids plus élevés aux attributs considérés comme moins importants pour la décision. En utilisant cette fonction de poids, le solveur retourne la seule raison suffisante de coût minimal, à savoir $x_1 \wedge x_3 \wedge x_4$, qui ne contient pas l'attribut non actionnable x_2. C'est donc l'explication préférée par le banquier pour cette instance particulière.*

7.2 L'impact des fonctions de poids sur les raisons abductives préférées

L'idée principale de cette section est d'aborder les variations dans les modalités d'agrégation des préférences des utilisateurs les attributs pour définir les raisons abductives préférées. Il est bien connu

que deux experts d'un même domaine peuvent avoir des préférences différentes. Cependant, en l'absence d'une application réelle avec de véritables préférences utilisateurs, notre étude se concentre sur l'exploration de différentes fonctions de poids, à la fois locales et globales.

Les fonctions de poids utilisées dans cette étude sont basées sur différentes approches telles que les valeurs de Shapley, LIME, Anchors, Explanatory, ainsi que Wordfreq et Feature importance. Ces fonctions de poids permettent de quantifier l'importance relative des différents attributs dans l'explication des résultats du modèle de classification. Il est possible de prendre en compte les préférences des utilisateurs, en attribuant des poids différents aux différents attributs en fonction de leur importance.

L'objectif de cette approche est de fournir des explications qui correspondent aux préférences des utilisateurs et qui sont cohérentes avec leur compréhension et leurs attentes. En utilisant différentes fonctions de poids, il est possible d'explorer différentes perspectives d'agrégation des préférences, ce qui peut conduire à des explications plus personnalisées et adaptées aux besoins individuels des utilisateurs.

7.2.1 Fonctions de poids

Fonctions de poids globales. Les fonctions de poids globales se concentrent sur la contribution des attributs en prenant en compte toutes les prédictions de toutes les instances. Dans ce paragraphe, nous allons présenter quelques-unes des mesures de poids globales utilisées dans la littérature pour définir des préférences utilisateurs.

— **Wordfreq :** la loi de Zipf stipule que la fréquence f d'un mot dans un corpus est inversement proportionnelle à son rang r, c-à-d que $f \propto \frac{1}{r}$. Cette loi est souvent utilisée pour modéliser la distribution des fréquences des mots dans un corpus linguistique. La fréquence Zipf f d'un mot est donnée par : $f = \log_{10}\left(\frac{N}{r}\right)$ où N est le nombre total de mots dans le corpus et r est le rang du mot, c'est-à-dire sa position dans le classement des mots les plus fréquents. https://pypi.org/project/wordfreq/

— **Features importance :** il s'agit de la méthode du « *Mean Decrease Impurity (MDI)* », qui permet d'évaluer l'importance des attributs sur une tâche de classification en mesurant la diminution moyenne de l'impureté (par exemple l'entropie ou l'indice Gini) dans l'arbre de décision lorsque l'attribut est utilisé pour diviser les données en sous-groupes. L'importance d'un attribut est alors évaluée en prenant la moyenne et l'écart type de cette diminution d'impureté sur l'ensemble des divisions de l'arbre qui utilisent cette attribut. https://scikit-learn.org/

Fonctions de poids locales. Les mesures locales se concentrent sur la contribution des attributs pour une prédiction spécifique, c'est-à-dire au niveau de chaque instance prédite. Nous présentons maintenant quelques mesures de poids locales

— **LIME (*Local Surrogate Models*) :** LIME permet d'expliquer les prédictions individuelles d'un modèle de Machine Learning non interprétable. Cette technique a été proposée et implémentée en 2016 [RSG16]. LIME se concentre sur la construction de modèles de substitution locaux pour expliquer les prédictions individuelles. L'idée est d'entraîner un modèle de substitution interprétable sur un nouveau jeu de données formé d'échantillons localement perturbés.

— **SHAP (*SHapley Additive exPlanations*) :** la valeur de Shapley est basée sur la théorie des jeux coopératifs. Le but de SHAP est d'expliquer la prédiction d'une observation en calculant la contribution de chaque variable à cette prédiction. Dans ce contexte, nous avons utilisé la méthode proposée par [LL17].

name	short description	(#instance #feature #class)	source
divorce	prédire si les couples divorceront ou non	(170, 54, 2)	Kaggle
compas	déterminer si les accusés récidiveront ou non sur une période de deux ans	(5278, 14, 2)	OpenML
employee	déterminer si les employés quitteront ou non l'entreprise au cours dans deux ans	(4653, 8, 2)	UCI
student mat	prédire si les étudiants réussiront ou échoueront en mathématiques	(395, 32, 2)	UCI
student por	prédire si les étudiants réussiront ou échoueront en portugais	(649, 32, 2)	UCI
anneal 2	prédire des informations sur le recuit, un traitement thermique utilisé en métallurgie	(898, 38, 2)	UCI
placement	prédire des informations sur le placement des étudiants	(215, 13, 2)	Kaggle
heart	prédire la présence ou l'absence de maladie cardiaque	(303, 13, 2)	OpenML
diabetes	prédire si les patients sont diabétiques ou non	(768, 8, 2)	Kaggle
horse	prédire si les chevaux peuvent survivre ou non	(299, 27, 2)	UCI
indian liver patient	classer les patients atteints de maladie du foie ou sans maladie	(583, 10, 2)	UCI
banknote	déterminer si les billets de banque sont authentiques ou non	(1372, 4, 2)	Kaggle
startup	prédire si les startups réussiront ou échoueront	(923, 45, 2)	Kaggle
farm-ads	décider si les propriétaires approuveront les publicités ou non	(4143, 54877, 2)	UCI

TABLE 7 – Certains datasets utilisés dans nos expériences.

7.2.2 Transformation monotone

Transformation monotone. La plupart du temps, les poids des attributs disponibles ne sont pas des entiers positifs. Cependant, les solveurs SAT, y compris celui que nous utilisons pour calculer les raisons suffisantes de coût minimal ne prennent en compte que des poids entiers positifs. Pour résoudre ce problème, nous appliquons une **transformation monotone croissante** pour convertir les poids en un vecteur de poids positifs. Nous appliquons la transformation suivante :

$$w \leftarrow w - \min_{x_i \in X_d} w(x_i)$$

Ensuite, nous multiplions le vecteur de poids w par 10^k, où k est le nombre maximal de chiffres après la virgule décimale puis nous ajoutons 1. Cette transformation permet de convertir tous les poids en entiers strictement positifs. Par exemple, si SHAP($\boldsymbol{x}, F$) donne $(\mathbf{0.5}, -\mathbf{0.2}, \mathbf{0.3}, -\mathbf{0.1})$, alors la transformation donne :

$$w(\boldsymbol{x}) = (\mathbf{8}, \mathbf{1}, \mathbf{6}, \mathbf{2})$$

Le poids de chaque attribut est donc rendu positif, mais l'ordre induit sur les explications est préservé de sorte que les raisons suffisantes de coût minimal ne changent pas. Lorsque des poids fractionnaires (mais positifs) sont pris en compte, une fois de plus, l'ordre induit sur les explications est préservé de sorte que les raisons suffisantes de coût minimal sont conservées. En effet, il est bien connu qu'une fonction d'utilité (ou de coût) peut être soumise à une transformation affine positive sans altérer l'ordre de préférence correspondant.

7.3 Résultats expérimentaux

Dans notre protocole expérimental, nous nous concentrons uniquement sur les raisons suffisantes de coût minimal.

7.3.1 Protocole expérimental

Nous avons sélectionné 18 jeux de données (datasets) pour la classification binaire, qui sont des jeux de données standards provenant des référentiels Kaggle (www.kaggle.com), OpenML (www.

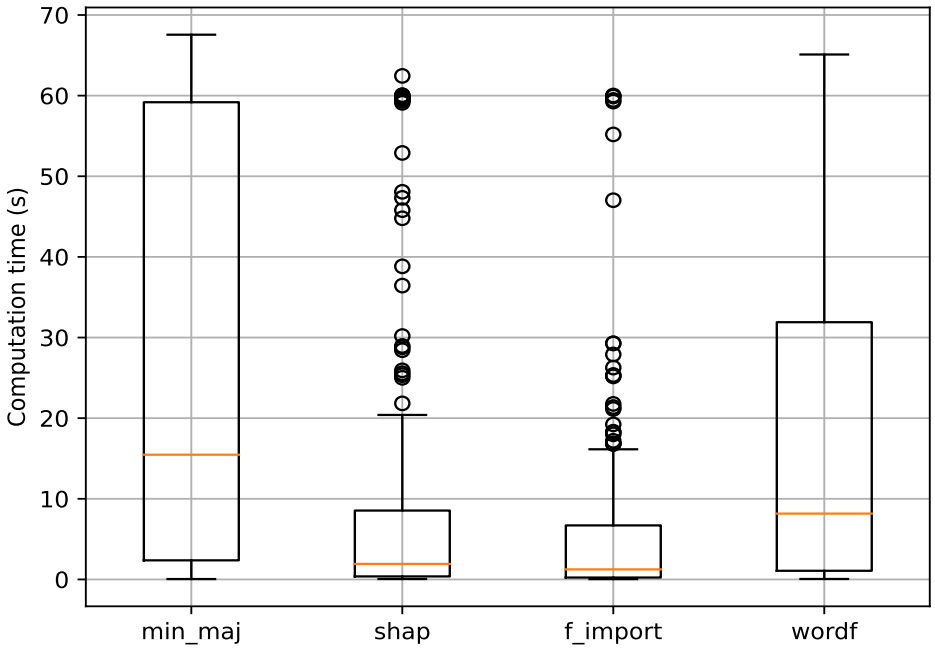


FIGURE 33 – Temps de calcul requis pour calculer toutes les raisons suffisantes minimales et les raisons suffisantes de coût minimal pour les fonctions de poids « SHAP », « f_importance » et « wordfreq » pour le *dataset* « *employee* ».

openml.org), et UCI (archive.ics.uci.edu/ml/). Pour chaque dataset, nous avons appris des arbres de décision et effectué une validation croisée à 10 blocs (10-*fold*). Certains de ces datasets sont répertoriés dans le tableau 7. « name » est le nom du benchmark. « short description » indique en quelques mots l'objectif de la prédiction. « #instance » est le nombre d'instances dans le dataset, « #feature » est le nombre d'attributs, et « #class » est le nombre de classes. Enfin, « source » précise l'origine du benchmark.

Lors de la phase d'apprentissage des arbres de décisions, les attributs catégoriels ont été traités comme des nombres arbitraires. Les attributs numériques ont été binarisés à la volée par l'algorithme d'apprentissage d'arbre de décision que nous avons utilisé, à savoir la version 0.23.2 de la bibliothèque Scikit-Learn [PVG+11]. Tous les hyper-paramètres de l'algorithme d'apprentissage ont été fixé à leur valeur par défaut. Pour chaque dataset b, un processus de validation croisée à 10 blocs a été réalisé. Pour chaque dataset b et chaque arbre de décision T pour b parmi les 10 qui ont été apprises, 25 instances ont été sélectionnées aléatoirement dans l'ensemble de test, ce qui a conduit à un ensemble de 250 instances par dataset (à moins que le dataset contient moins de 250 instances, auquel cas le pool est constitué de l'ensemble des instances complet du dataset).

Dans nos expériences, nous nous sommes concentrés sur le calcul des raisons suffisantes de coût minimal. Pour chaque dataset b, nous avons généré toutes les raisons suffisantes de coût minimal pour les instances $\boldsymbol{x}$ dans le pool associé à b, jusqu'à ce que nous atteignions le nombre maximal ou dépassions la limite de temps (TO) de 60 secondes par instance. Chaque fois qu'une raison suffisante de coût minimal pour l'instance $\boldsymbol{x}$ a été générée, cette raison a été bloquée pour éviter qu'elle soit calculée une deuxième fois.

Étant donné qu'aucune fonction de poids n'est associée aux datasets utilisés dans nos expériences (rappelons que les préférences de l'utilisateur ne sont pas intrinsèques aux jeux de données), nous avons considéré pour chaque dataset les fonctions de poids suivantes (1), la fonction de poids uniforme w_1, où chaque attribut a un poids égal à 1 (dans ce cas, les raisons suffisantes de coût minimal sont précisément les raisons suffisantes minimales), (2) pour chaque arbre de décision T, le score SHAP opposé [LL17, LEC+20] de chaque attribut de l'instance $\boldsymbol{x}$ étant donné T calculé à l'aide de la méthode SHAP (`shap.readthedocs.io/en/latest/api.html`), (3) le score d'importance (*feature importance*) f opposé de chaque attribut étant donnée T a été calculé à l'aide de la bibliothèque Scikit-Learn [PVG+11], (4) le score de fréquence Zipf opposé de chaque attribut qui est considéré comme un mot dans la bibliothèque *wordfreq*. Dans le cas (2), les poids calculés sont des poids locaux, c'est-à-dire qu'ils dépendent de l'instance $\boldsymbol{x}$ considérée. Cela contraste avec les poids calculés dans les cas (3) (mesurant l'importance globale des attributs) et (4). Alors que (2) et (3) visent à favoriser les raisons suffisantes impliquant les attributs les plus importants du point de vue de l'explicabilité, (4) met l'accent sur l'intelligibilité.

Nous avons considéré un temps limite de génération de 60 secondes par instance. Étant donné que les fonctions de poids font partie de l'entrée, nous n'avons pas compté le temps de calcul nécessaire pour les générer dans les 60 secondes. Nous avons utilisé le solveur WEIGHTED PARTIAL MAXSAT `openwbo` [MML14] pour calculer raisons suffisantes de coût minimal via l'approche présentée à la proposition 24. Toutes les expériences ont été conduites sur un ordinateur équipé de Processeur Intel(R) Core(TM) $i9-99003,10$ GHz - 16 cœurs et 64 Go de mémoire.

7.3.2 Résultats expérimentaux

Le tableau 8 présente un extrait des résultats, basé sur 18 datasets. Pour chaque dataset et les arbres de décision appris à partir de celui-ci, le tableau donne le nom du dataset (nom), la précision moyenne (%A) des 10 arbres, le nombre moyen d'attributs binaires (#B) dans ceux-ci, et le nombre d'instances du dataset (#I). Ensuite, pour chaque dataset b et chaque fonction de poids utilisée, il indique le nombre d'instances $\boldsymbol{x}$ dans le pool pour lesquelles au moins une (1) ou toutes (A) les raisons suffisantes de coût minimal pour $\boldsymbol{x}$ ont été dérivées en moins de 60 secondes, la colonne (nb) donne le nombre moyen (et l'écart-type) de raisons suffisantes de coût minimal qui ont été obtenues pour les instances pour lesquelles toutes les raisons suffisantes de coût minimal ont pu être calculées.

Les résultats empiriques montrent clairement que le calcul des raisons suffisantes préférées (de coût minimal) est réalisable en pratique. En effet, pour de nombreux datasets et instances, toutes les raisons suffisantes de coût minimal ont été calculées en moins de 60 secondes, quel que soit le type de fonction de poids utilisé. Nous pouvons également observer que le nombre d'instances pour lesquelles toutes les raisons suffisantes de coût minimal ont été calculées est souvent proche du nombre d'instances pour lesquelles au moins une raison suffisante de coût minimal a été dérivée dans le délai imparti. De plus, nous avons pu observer que l'utilisation des types de fonctions de poids (2), (3), et (4) a eu un effet radical sur le nombre de raisons, favorisant ainsi le calcul de toutes les raisons suffisantes de coût minimal en réduisant leur nombre. Enfin, il convient de noter que pour chaque dataset b (y compris les « plus difficiles », par exemple, *farm-ads*), chaque instance dans le pool de b, et chaque type de fonction de poids (1) à (4), il a été possible de calculer en moins de 60 secondes une raison suffisante de coût minimal. Et même dans de nombreux de cas, il a été possible d'énumérer presque toutes les raisons suffisantes préférées en moins de 60 secondes, comme le montre la figure 33, dans le cas du benchmark « *employee* ». Des statistiques plus détaillées sur le temps de calcul, le nombre de raisons générées et leur taille ont été établies pour chaque dataset elles sont disponibles depuis `www.cril.univ-artois.fr/expekctation/`.

dataset / decision tree				minimum-size			SHAP			f-importance			wordfreq		
name	%A	#B	#I	l	A	nb	l	A	nb	l	A	nb	l	A	nb
divorce	97.06	3	170	170	170	1.58 (±1.1)	170	170	1.2 (±0.6)	170	170	1.4 (±0.4)	170	170	1.65 (±0.9)
compas	66.38	40	250	250	250	3.34 (±2.9)	250	248	1.7 (±1.1)	250	244	3.7 (±2.5)	250	246	2.7 (±2.1)
employee	82.76	65	250	250	248	4.7 (±4.6)	250	247	2.2 (±2.1)	250	249	2.1 (±2.5)	248	250	3.4 (±2.6)
student mat	89.23	23	250	250	217	2.85 (±2.1)	250	250	2.3 (±1.5)	250	250	2.8 (±1.7)	250	250	2.2 (±1.4)
student por	91.79	31	250	250	247	2.85 (± 1.8)	250	248	2.2 (±1.4)	250	249	2.9 (±1.8)	250	248	2.6 (±1.7)
anneal	99.11	13	250	250	250	1.28 (±0.5)	250	250	3.1 (±2.3)	250	250	3.3 (±2.4)	250	250	4.1 (±2.8)
placement	89.55	17	215	215	215	5.7 (±6)	215	215	1.7 (±1.3)	215	215	2.4 (±2.7)	215	215	2.2 (±1.8)
diabetes	72.78	110	250	250	248	4.2 (±4.7)	250	250	2.3 (±2.1)	250	250	2.8 (±2.1)	250	250	2.2 (±1.9)
horse	75.6	40	250	235	6	6.2 (±7.8)	250	235	4.6 (±5.9)	250	240	4.4 (±3.8)	250	238	4.7 (±5.6)
ind. l. pat.	64.61	86	250	250	250	2.4 (±1.3)	250	250	1.6 (±1.4)	250	250	1.5 (±0.9)	250	250	2.3 (±2.8)
startup	70.8	99	250	148	14	48.2 (49.5)	162	145	1.4 (±0.9)	133	142	1.7 (±1.2)	163	138	1.8 (±1.1)
dermatology	98.2	7	250	250	250	2.1 (±1.2)	250	250	1.3 (±0.4)	250	250	1.4 (±0.6)	250	250	1.4 (±0.2)
hepatitis	82.74	13	142	142	142	2.6 (±1.9)	142	142	1.8 (±1.5)	142	142	2.1 (±1.8)	142	142	2.3 (±2.1)
vote house 84	94.66	15	250	250	250	1.9 (±2.2)	250	250	1.1 (±0.3)	250	250	1.0 (±0.2)	250	250	1.0 (±0.2)
farm-ads	80.78	328	250	250	0	- (-)	36	31	2.2 (±2.5)	26	20	3.2 (±2.1)	41	29	3.3 (±2.6)
mnist49	95.99	287	250	250	0	- (-)	46	38	4.2 (±3.4)	40	38	5.4 (±4.8)	42	41	5.3 (±4.1)
mnist38	95.99	287	250	250	51	34.4 (38.1)	87	48	3.2 (±2.7)	76	48	4.1 (±2.9)	87	48	3.2 (±2.7)
adult	81.16	2433	250	145	142	3.6 (±4.2)	143	138	2.6 (±2.2)	144	140	3.1 (±2.8)	137	133	2.9 (±2.6)

TABLE 8 – Quelques statistiques sur le calcul des raisons suffisantes de coût minimal.

7.4 Conclusion

Dans ce chapitre, nous avons examiné le problème de la génération d'explications abductives préférées. Il s'avère qu'une instance peut avoir un nombre exponentiel d'explications abductives, voire un nombre exponentiel d'explications abductives de taille minimale.

Nous avons présenté des modèles de préférence, analysé la complexité de calculer une explication abductive préférée pour les arbres de décision. Dans chaque cas, nous avons présenté une approche permettant de calculer de telles explications. Au-delà de conduire à des explications qui correspondent mieux aux attentes de l'utilisateur, les expériences ont montré que l'exploitation des préférences de l'utilisateur peut réduire considérablement le nombre d'explications, rendant leur énumération possible dans des situations où le calcul de toutes les raisons suffisantes serait irréalisable.

L'utilisation des préférences de l'utilisateur, comme illustré dans ce travail, représente une première étape vers la caractérisation des explications qui correspondent aux attentes de l'utilisateur. Cependant, il est important de noter que l'évaluation des explications reste un défi complexe. Pour effectuer une évaluation significative [DK17], il serait idéal qu'un expert humain soit disponible. Malheureusement, dans la plupart des cas, un tel expert est difficile à obtenir. Il est à noter qu'à ce jour, il n'existe pas de modèles d'utilisateur sophistiqués qui puissent remplacer un expert humain. Cela souligne le besoin de continuer à travailler sur le développement de modèles d'utilisateur avancés pour faciliter l'évaluation des explications. Un aspect essentiel à discuter pour améliorer notre approche réside dans l'interaction entre l'expert et le modèle de préférence. Cette interaction joue, en effet, un rôle fondamental dans la détermination de bonnes explications.

8

Approximation des explications probabilistes par minimisation super-modulaire

Sommaire

8.1 Introduction . . . 137
8.2 Explication probabiliste . . . 138
8.2.1 Explication probabiliste . . . 138
8.2.2 Formulation du problème . . . 139
8.2.3 Évaluation des erreurs d'explication . . . 140
8.3 Minimisation super-modulaire . . . 141
8.3.1 Minimisation de l'erreur d'explication . . . 142
8.3.2 Minimisation de l'erreur d'explication non normalisée . . . 143
8.4 Algorithmes approchés . . . 144
8.4.1 Descente gloutonne (GD) . . . 144
8.4.2 Ascension gloutonne (GA) . . . 145
8.4.3 Application aux arbres de décision . . . 147
8.5 Expérimentations . . . 148
8.5.1 Protocole expérimental . . . 148
8.5.2 Résultats . . . 150
8.5.3 Expérimentations supplémentaires . . . 150
8.6 Discussion . . . 153
8.7 Conclusion . . . 154

Expliquer de manière précise et intelligible les prédictions réalisées par des classifieurs est un défi clé en XAI. Dans cette optique, calculer une explication abductive d'une instance vise à expliquer la prédiction faite. Cependant, vu nos limitations cognitives, les explications abductives sont parfois de trop grande taille pour être interprétables. Lorsque cela se produit, nous devons réduire la taille des explications tout en déterminant toujours la classe prédite avec une probabilité élevée. Dans ce travail, nous montrons que calculer de telles « explications probabilistes » est NP-difficile, même pour la classe restreinte des arbres de décision.

Afin de contourner le problème, nous étudions dans ce chapitre l'approximation des explications probabilistes sous l'angle de la super-modularité. Nous examinons à la fois les approches de descente

gloutonne (GD) et d'ascension gloutonne (GA) pour la minimisation super-modulaire, dont les garanties d'approximation dépendent de la courbure de la fonction d'erreur « non normalisée » qui évalue la précision de l'explication. Basés sur diverses expériences visant à expliquer les prédictions des arbres de décision, nous montrons que nos algorithmes gloutons offrent une alternative efficace à la méthode exacte basé sur un encodage SAT.

Une partie de ces travaux a donné lieu à la publication [BK23b]. Nous tenons à signaler que dans cette publication, les preuves des propositions 4 et 5 contenaient des erreurs qui ont été corrigées dans le manuscrit de thèse. Nous précisons également qu'une version anglaise corrigée de [BK23b] a été déposée sur HAL [BK23a].

8.1 Introduction

Parmi les divers types d'explications proposés dans la littérature XAI, les explications « formelles » sont particulièrement intéressantes, car leur validité peut être mathématiquement prouvée [IISMS22].

Cependant, la validité des explications n'est pas le seul critère à prendre en compte pour obtenir explications utilisables. La propriété de « concision » est également importante, car une explication abductive avec trop d'attributs ne peut pas être comprise par les utilisateurs humains. En effet, en psychologie cognitive, il est depuis longtemps reconnu qu'il existe une limite supérieure à notre capacité de raisonnement sur des éléments interagissant simultanément. Comme l'a conjecturé [Mil56], cette limite est de $sept$ éléments plus ou moins $deux$, et depuis lors, elle a été confirmée par de nombreuses expériences en sciences cognitives. Ainsi, restreindre la taille des explications apparaît comme nécessaire pour garantir leur intelligibilité.

Sur la base de ces considérations, comment pouvons-nous réduire la taille des explications tout en préservant une grande partie leur validité ? C'est là que les *explications probabilistes* [WMHK21] entrent en jeu. Supposons que nous avons accès à une explication abductive I pour un classifieur h et une instance $\boldsymbol{x}$, ainsi que d'une limite de taille $k \leq |I|$. Pour tout sous-ensemble candidat S de I, soit $\epsilon_{h,\boldsymbol{x}}(S)$ la probabilité qu'une instance aléatoire couverte par $\boldsymbol{x}_S$ [16] soit classée différemment de $\boldsymbol{x}$ par le classifieur h. En d'autres termes, $\epsilon_{h,\boldsymbol{x}}(S)$ est la probabilité de commettre une « erreur d'explication » pour déduire $h(\boldsymbol{x})$ en utilisant $\boldsymbol{x}_S$ au lieu de $\boldsymbol{x}$. Sur cette base, le principal problème examiné dans la suite de ce chapitre est de trouver une explication probabiliste $S \subseteq I$ de taille au plus k de telle sorte que $\epsilon_{h,\boldsymbol{x}}(S)$ soit minimisé. Malheureusement, cette tâche d'optimisation est très coûteuse du point de vue computationnel. En effet, le problème de trouver un sous-ensemble S qui minimise $\epsilon_{h,\boldsymbol{x}}(\cdot)$ sous la contrainte de cardinalité $|S| \leq k$ est $\mathsf{NP}^{\mathsf{PP}}$-difficile pour les classifieurs d'une manière générale [WMHK21], et NP-difficile pour les arbres de décision [ABROS22]. Comme le montre la présente étude, ce problème reste NP-difficile pour les arbres de décision même dans le cas restreint où S est un sous-ensemble d'une raison suffisante donnée I.

Afin de surmonter une telle barrière calculatoire, nous étudions *l'approximation* des explications probabilistes à travers le prisme de la super-modularité. Comme $\epsilon_{h,\boldsymbol{x}}(S)$ peut être considéré comme le nombre $\mu_{h,\boldsymbol{x}}(S)$ d'erreurs induites par le choix de S, en moyenne sur le nombre d'instances couvertes par $\boldsymbol{x}_S$, nos résultats exploitent deux propriétés clés : (i) la fonction d'erreur *non normalisée* $\mu_{h,\boldsymbol{x}}(\cdot)$ est super-modulaire et monotone décroissante, et (ii) le facteur de normalisation est *constant* pour tous les sous-ensembles S de même taille. Ainsi, même si $\epsilon_{h,\boldsymbol{x}}(\cdot)$ n'est pas super-modulaire, nous pouvons toujours utiliser des algorithmes d'approximation pour la minimisation super-modulaire, en les couplant

16. La restriction de $\boldsymbol{x}$ à S, notée $\boldsymbol{x}_S$, est l'instance partielle dans $\{0, 1, *\}^d$ telle que, pour chaque $i \in [d]$, $(x_S)_i = x_i$ si $i \in S$, et $(x_S)_i = *$ sinon.

avec une méthode de sélection par niveaux, afin de dériver des explications probabilistes dotées de *garanties mathématiques d'approximation*.

Dans ce travail, nous présentons principalement deux algorithmes conceptuellement simples et faciles à implémenter, dont les facteurs d'approximation dépendent de la courbure c de la fonction d'erreur non normalisée $\mu_{h,\boldsymbol{x}}(\cdot)$. Le premier algorithme est une *descente gloutonne* GD qui démarre de la raison suffisante I et qui, à chaque itération, élimine de la solution courante S un attribut i tel que le nombre d'erreurs $\mu_{h,\boldsymbol{x}}(S \setminus \{i\})$ est minimal. Le second algorithme est une *ascension gloutonne* GA qui part de l'ensemble vide $\varnothing$ et qui, à chaque itération, ajoute à la solution courante S un attribut i dont le nombre d'erreurs $\mu_{h,\boldsymbol{x}}(S \cup \{i\})$ est minimal. Pour ces deux algorithmes, une étape supplémentaire de minimisation d'erreur $\epsilon_{h,\boldsymbol{x}}$ sur la chaîne (descente ou ascendante) des solutions parcourues par la méthode gloutonne est effectuée, afin de dériver la meilleure de ces solutions. Les facteurs d'approximation pour ces deux algorithmes dépendent de la courbure c. Les tailles des explications retournées par la descente gloutonne et l'ascension gloutonne sont respectivement bornées par k et $k \ln\left(\frac{2e}{1-c}\right)$.

Les deux algorithmes sont comparés empiriquement à l'approche exacte basée sur un encodage SAT suggérée dans [ABROS22], qui vise à inférer des explications probabilistes optimales pour les arbres de décision. Les résultats expérimentaux indiquent que nos algorithmes gloutons peuvent trouver efficacement des explications précises et, contrairement à l'approche basée sur SAT (appelée dans le reste de ce chapitre, méthode SAT), ils sont capables de s'adapter à des tâches d'explication de grande dimension.

8.2 Explication probabiliste

Dans cette section, nous débutons par fournir quelques notions de base sur les explications probabilistes, puis nous examinons certains aspects computationnels liés à leur évaluation et à leur optimisation.

8.2.1 Explication probabiliste

Nous rappelons que pour un entier positif d, nous utilisons $[d]$ pour désigner l'ensemble $\{1, \cdots, d\}$. Les classifieurs considérés dans cette étude sont des fonctionnes booléennes de la forme $h : \{0,1\}^d \to \{0,1\}$. Une instance partielle est un vecteur $\boldsymbol{z} \in \{0,1,*\}^d$, où $z_i = *$ indique que la i-ème attribut de $\boldsymbol{z}$ est non définie. Une instance $\boldsymbol{x}$ est couverte par $\boldsymbol{z}$ si $x_i = z_i$ pour tous les attributs $i \in [d]$ telles que $z_i \neq *$. Pour un sous-ensemble $S \subseteq [d]$ vu comme une raison possible de la classification de $\boldsymbol{x}$ étant donné h, la restriction de $\boldsymbol{x}$ à S, notée $\boldsymbol{x}_S$, est l'instance partielle dans $\{0,1,*\}^d$ telle que, pour chaque $i \in [d]$, $(x_S)_i = x_i$ si $i \in S$, et $(x_S)_i = *$ sinon. Clairement, une instance $\boldsymbol{y} \in \{0,1\}^d$ est couverte par $\boldsymbol{x}_S$ si et seulement si $\boldsymbol{y}_S = \boldsymbol{x}_S$.

Remarque 13. *Dans ce chapitre, nous faisons référence à la raison suffisante d'un ensemble donné I comme étant le terme associé t_I. Par exemple, pour l'instance $\boldsymbol{x} = (0,1,1)$ et un classifieur h, on dit que $S = \{1,3\}$ est une raison suffisante pour $\boldsymbol{x}$ étant donné h si seulement et si que la raison suffisante est $t_{\{1,3\}} = \neg x_1 \wedge x_3 = \{\neg x_1, x_3\}$.*

Étant donné un classifieur h, et une instance $\boldsymbol{x}$ pour laquelle la prédiction $h(\boldsymbol{x})$ doit être expliquée, soit $\epsilon_{h,\boldsymbol{x}} : 2^{[d]} \to \mathbb{R}$ la *fonction d'erreur* définie par

$$\epsilon_{h,\boldsymbol{x}}(S) = \frac{|\{\boldsymbol{y} \in \{0,1\}^d : h(\boldsymbol{y}) \neq h(\boldsymbol{x}), \boldsymbol{y}_S = \boldsymbol{x}_S\}|}{|\{\boldsymbol{y} \in \{0,1\}^d : \boldsymbol{y}_S = \boldsymbol{x}_S\}|} \tag{5}$$

Comme indiqué ci-dessus, $\epsilon_{h,\boldsymbol{x}}(S)$ peut être interprété comme la probabilité de commettre une « erreur d'explication » en utilisant l'instance partielle $\boldsymbol{x}_S$ plutôt que l'instance complète $\boldsymbol{x}$. Étant donné

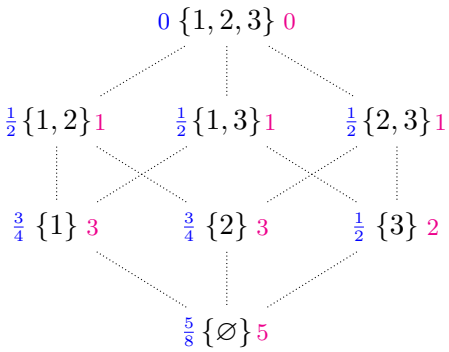


FIGURE 34 – L'erreur $\epsilon_{h,\boldsymbol{x}}(S)$ (en bleu) et le nombre d'erreurs $\mu_{h,\boldsymbol{x}}(S)$ (en magenta) pour chaque $S \subseteq [3]$, en utilisant le classifieur h donné par (8) et l'instance $\boldsymbol{x} = (1, 1, 1)$.

un paramètre de précision $\varepsilon \in [0, 1]$, une explication S est appelée $(1 - \varepsilon)$-probable si $\epsilon_{h,\boldsymbol{x}}(S) \leq \varepsilon$ [WMHK19]. Formellement,

Définition 67 (raison δ-probable). *Soit h un classifieur booléen et $\boldsymbol{x} \in \{0, 1\}^d$ tel que $h(\boldsymbol{x}) = 1$ et $\delta \in [0, 1]$. Une raison S est* δ-probable *pour $\boldsymbol{x}$ étant donné h si $1 - \epsilon_{h,\boldsymbol{x}}(S) \geq \delta$.*

Nous rappelons également que S est une raison suffisante pour $\boldsymbol{x}$ étant donné h si $\epsilon_{h,\boldsymbol{x}}(S) = 0$, et $\epsilon_{h,\boldsymbol{x}}(S') > 0$ pour tout sous-ensemble propre S' de S. Notons que (5) peut être réécrit comme

$$\epsilon_{h,\boldsymbol{x}}(S) = \frac{\mu_{h,\boldsymbol{x}}(S)}{2^{d-|S|}} \tag{6}$$

où $\mu_{h,\boldsymbol{x}}(S)$ est le nombre d'erreurs induites par le choix de S, c'est-à-dire,

$$\mu_{h,\boldsymbol{x}}(S) = |\{\boldsymbol{y} \in \{0, 1\}^d : h(\boldsymbol{y}) \neq h(\boldsymbol{x}), \boldsymbol{y}_S = \boldsymbol{x}_S\}| \tag{7}$$

Exemple 43. *Considérons le classifieur $h : \{0, 1\}^3 \to \{0, 1\}$ spécifié par la fonction polynomiale à seuil suivante :*

$$h(\boldsymbol{x}) = 1 \Leftrightarrow x_1x_2x_3 + x_1x_2 - x_1 - x_2 \geq 0 \tag{8}$$

Étant donné l'instance $\boldsymbol{x} = (1, 1, 1)$ pour laquelle nous devons expliquer $h(\boldsymbol{x}) = 1$, et en utilisant le diagramme de Hasse dans la figure 34, on observe que le terme associé à l'ensemble $\{1, 2, 3\}$, à savoir $t_{\{1,2,3\}} = x_1 \wedge x_2 \wedge x_3$ est la seule raison suffisante pour $\boldsymbol{x}$ étant donné h. Cependant, les termes associés aux deux ensembles $\{1, 2\}$ et $\{3\}$ sont tous les deux des explications $\frac{1}{2}$-probables minimales pour l'inclusion ensembliste pour $\boldsymbol{x}$ étant donné h.

8.2.2 Formulation du problème

Le principal problème examiné dans la suite est le suivant :

Problème 1. *Étant donné un classifieur $h : \{0, 1\}^d \to \{0, 1\}$, une instance $\boldsymbol{x} \in \{0, 1\}^d$, un ensemble $I \subseteq [d]$, une limite de taille $k \leq |I|$, trouver un sous-ensemble $S \subseteq I$ de taille au plus k tel que $\epsilon_{h,\boldsymbol{x}}(S)$ soit minimisé.*

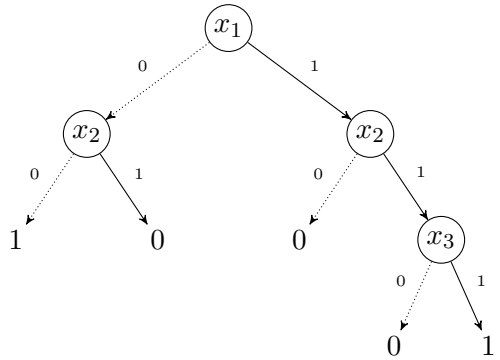


FIGURE 35 – Une représentation par arbre de décision de (8).

Notons que pour répondre au problème 1, il est nécessaire d'évaluer la fonction $\epsilon_{h,x}$ qui mesure l'erreur de prédiction du classifieur sur l'ensemble des instances couvertes par x_S. Pour résoudre le problème 1, nous devons essayer différentes combinaisons de sous-ensembles d'attributs, évaluer la fonction d'erreur pour chaque combinaison, et choisir une de celles qui donnent la valeur minimale de la fonction d'erreur. En résumé, l'évaluation de la fonction d'erreur est cruciale pour déterminer la qualité des différentes sélections d'attributs et pour identifier celle qui répond le mieux au problème en minimisant l'erreur de classification.

Exemple 44. *Considérons le classifieur h décrit à l'exemple 8. Pour l'instance $\boldsymbol{x} = (1, 1, 1)$, nous avons déjà observé que le terme associé à l'ensemble $\{1, 2, 3\}$, c'est-à-dire $t_{\{1,2,3\}} = x_1 \wedge x_2 \wedge x_3$, est la seule raison suffisante pour $\boldsymbol{x}$ étant donné h. À présent, nous cherchons à trouver une explication d'une taille au plus $k = 2$ qui minimise l'erreur de classification. Nous pouvons constater à partir du diagramme de Hasse 34 que les ensembles $\{3\}, \{1, 3\}, \{2, 3\}, \{1, 2\}$ sont des solutions du problème d'optimisation 1, compte tenu du classifieur h, de l'instance x, de la contrainte de cardinalité $k = 2$, et de $I = [3]$. En revanche, pour $k = 1$, l'ensemble $\{3\}$ est la seule solution du problème d'optimisation 1.*

8.2.3 Évaluation des erreurs d'explication

Il est facile de voir que le problème d'évaluer $\epsilon_{h,\boldsymbol{x}}(S)$ pour un classifieur arbitraire est en général #P-difficile [17]. Cependant, [IHI+22] ont montré que $\epsilon_{h,\boldsymbol{x}}(S)$ peut être calculé en temps polynomial lorsque h est décrit par un arbre de décision. Pour plus de clarté, nous montrons ici que $\epsilon_{h,\boldsymbol{x}}(S)$ peut être évalué en temps linéaire pour les arbres de décision, en utilisant l'orthogonalité des arbres de décision et le fait que cette propriété est préservée sous conditionnement.

Nous rappelons qu'un terme est contradictoire s'il inclut une paire $\{l, \neg l\}$ de littéraux opposés. Une formule DNF $\phi = \{t_1, \cdots, t_m\}$ est orthogonale si $t_i \cup t_j$ est contradictoire pour toutes les paires $i, j \in [m]$ telles que $i \neq j$. Le conditionnement [Dar99] de ϕ par un terme t, noté $\phi \mid t$, est la formule obtenue en supprimant de $\{t_1 \cup t, \cdots, t_m \cup t\}$ tout terme qui est contradictoire.

Proposition 25. *Soit un classifieur $h : \{0,1\}^d \to \{0,1\}$ représenté par un arbre de décision $\mathcal{T}$, une instance $\boldsymbol{x} \in \{0,1\}^d$, et un sous-ensemble d'attributs $S \subseteq [d]$, évaluer $\epsilon_{h,\boldsymbol{x}}(S)$ peut être réalisé en temps $\mathcal{O}(|S| \cdot |\mathcal{T}|)$.*

Preuve 7. *Nous avons déjà rappelé dans le chapitre 6 qu'un arbre de décision $\mathcal{T}$ peut être transformé en temps linéaire en une formule DNF orthogonale équivalente, notée $\mathtt{DNF}(\mathcal{T})$, où chaque terme correspond*

17. En effet, lorsque h est représenté par une formule en forme normale conjonctive, évaluer $\epsilon_{h,x}(S)$ pour $S = \varnothing$ est équivalent à résoudre le problème #SAT.

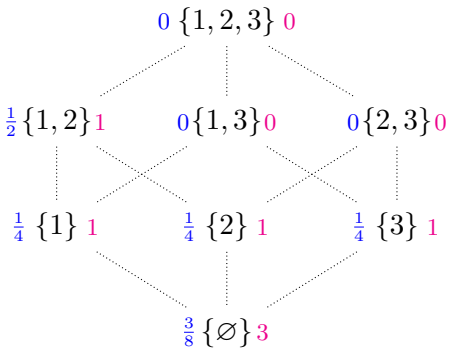


FIGURE 36 – L'erreur $\epsilon_{h,\boldsymbol{x}'}(S)$ (en bleu) et le nombre d'erreurs $\mu_{h,\boldsymbol{x}'}(S)$ (en magenta) pour chaque $S \subseteq [3]$, en utilisant le classifieur h donné par (8) et l'instance $\boldsymbol{x}' = (1,1,0)$ $(h(\boldsymbol{x}') = 0)$.

à un chemin de la racine à une feuille étiquetée par 1. *Étant donné une instance* $\boldsymbol{x} \in \{0,1\}^d$ *et un ensemble* S *d'attributs, soit* $t_{\boldsymbol{x}_S}$ *le terme associé à l'instance partielle* $\boldsymbol{x}_S$*, c'est-à-dire,*

$$t_{\boldsymbol{x}_S} = \bigcup_{i=1}^{d} \{x_i : (x_S)_i = 1\} \cup \{\overline{x}_i : (x_S)_i = 0\}$$

par construction, $\texttt{DNF}(\mathcal{T}) \mid t_{\boldsymbol{x}_S}$ *est orthogonale et donc, pour les arbres de décision, (7) peut simplement être reformulé comme suit :*

$$\mu_{h,\boldsymbol{x}}(S) = \begin{cases} \sum_{t \in \texttt{DNF}(\mathcal{T}) \mid t_{\boldsymbol{x}_S}} 2^{d-|t|} & \text{if } h(\boldsymbol{x}) = 1 \\ 2^d - \sum_{t \in \texttt{DNF}(\mathcal{T}) \mid t_{\boldsymbol{x}_S}} 2^{d-|t|} & \text{if } h(\boldsymbol{x}) = 0 \end{cases}$$

Le résultat découle alors de (6), en conjonction avec le fait que $\texttt{DNF}(\mathcal{T}) \mid t_{\boldsymbol{x}_S}$ *peut être dérivé en temps* $\mathcal{O}(|S| \cdot |\mathcal{T}|)$.

8.3 Minimisation super-modulaire

Dans cette section, nous démontrons que le problème 1 est NP-difficile lorsque le classifieur h est représenté par un arbre de décision. Ce résultat confirme la complexité intrinsèque du problème tel que défini dans la section précédente. Nous montrons que contrairement à la fonction d'erreur $\epsilon_{h,\boldsymbol{x}}(\cdot)$, qui n'est ni monotone ni super-modulaire, la fonction d'erreur non normalisée $\mu_{h,\boldsymbol{x}}(\cdot)$ présente des propriétés favorables. Elle est super-modulaire, monotone décroissante, et positive. Ces propriétés sont cruciales pour l'étude de l'approximation du problème 1.

Exemple 45. *Considérons le classifieur* h *décrit à l'exemple 8. Pour l'instance* $\boldsymbol{x} = (1,1,1)$*, et comme le montre le diagramme de Hasse de la figure 34, la fonction d'erreur* $\epsilon_{h,\boldsymbol{x}}(.)$ *n'est pas super-modulaire. Pour les deux ensembles* $S = \{1\}$ *et* $T = \{2\}$*, nous avons* $\epsilon_{h,\boldsymbol{x}}(S) + \epsilon_{h,\boldsymbol{x}}(T) = \frac{3}{2} > \frac{9}{8} = \epsilon_{h,\boldsymbol{x}}(S \cup T) + \epsilon_{h,\boldsymbol{x}}(S \cap T)$*. D'un autre côté, pour l'instance* $\boldsymbol{x}' = (1,1,0)$ $(\mathcal{T}(\boldsymbol{x}') = 0)$*, le diagramme de Hasse de la figure 36 montre que la fonction n'est pas sous-modulaire. Pour les deux ensembles* $S = \{1\}$ *et* $T = \{2\}$*, nous avons* $\epsilon_{h,\boldsymbol{x}'}(S) + \epsilon_{h,\boldsymbol{x}'}(T) = \frac{1}{4} + \frac{1}{4} = \frac{1}{2} < \frac{3}{8} + \frac{1}{2} = \frac{7}{8} = \epsilon_{h,\boldsymbol{x}'}(S \cup T) + \epsilon_{h,\boldsymbol{x}'}(S \cap T)$*. Par conséquent, nous pouvons conclure que la fonction d'erreur* $\epsilon_{h,\boldsymbol{x}'}$ *n'est ni super-modulaire ni sous-modulaire, ni monotone décroissante ni monotone croissante.*

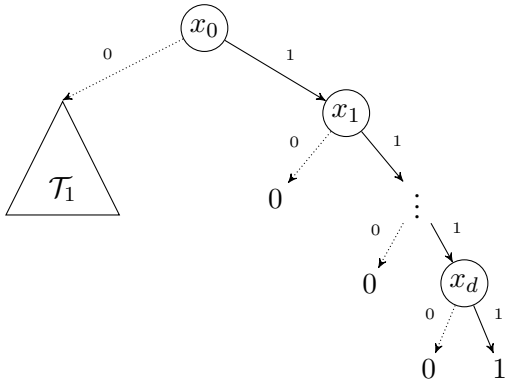


FIGURE 37 – L'arbre de décision $\mathcal{T}_2$ utilisé dans la preuve de la proposition 26.

8.3.1 Minimisation de l'erreur d'explication

Nous présentons maintenant un résultat qui montre que la version décisionnelle du problème 1 est généralement difficile à résoudre pour les arbres de décision, même lorsque l'ensemble I considéré en entrée est une raison suffisante.

Proposition 26. *Soit un classifieur h représenté par un arbre de décision $\mathcal{T}$, une instance $\boldsymbol{x} \in \{0,1\}^d$, et une raisons suffisante $I \subseteq [d]$ pour $\boldsymbol{x}$ étant donné h, un entier $k \leq |I|$, et un seuil $\varepsilon \in (0,1)$. Le problème de déterminer un sous-ensemble $S \subseteq I$ d'une taille au plus k qui satisfait $\epsilon_{h,\boldsymbol{x}}(S) \leq \varepsilon$ est* NP*-difficile.*

Preuve 8. *Considérons le problème $\boldsymbol{P}_2$ défini ci-dessus. Notre objectif est de donner une réduction en temps polynomial de $\boldsymbol{P}_1$ vers $\boldsymbol{P}_2$, où $\boldsymbol{P}_1$ est le problème suivant :*

> *Étant donné un classifieur h représenté par un arbre de décision $\mathcal{T}$, une instance $\boldsymbol{x} \in \{0,1\}^d$, et un entier $k \leq d$, le problème est de déterminer un sous-ensemble $S \subseteq [d]$ tel que $|S| \leq k$ et $\epsilon_{h,\boldsymbol{x}}(S) = 0$.*

Comme montré dans [BMPS20], ce problème est NP*-difficile. Considérons une instance $(\mathcal{T}_1, \boldsymbol{x}_1, k_1)$ de $\boldsymbol{P}_1$, et soit h_1 l'hypothèse associée à $\mathcal{T}_1$. Sans perte de généralité, nous supposons que $\boldsymbol{x}_1$ est le vecteur $\mathbf{1}$ de dimension d, noté $\mathbf{1}$, et que $h_1(\boldsymbol{x}_1) = 1$. Nous construisons une instance $(\mathcal{T}_2, \boldsymbol{x}_2, I, k_2, \varepsilon)$ de $\boldsymbol{P}_2$ de la manière suivante. L'arbre de décision $\mathcal{T}_2$ est défini selon la figure 37. Plus précisément, le nœud racine de $\mathcal{T}_2$ est étiqueté avec x_0, le sous-arbre enraciné dans le fils gauche de x_0 est l'arbre de décision $\mathcal{T}_1$, et le sous-arbre enraciné dans le fils droit de x_0 est la chenille qui encode la conjonction $x_1 \wedge \cdots \wedge x_d$. Les composants restants de l'instance du problème sont donnés comme suit : $\boldsymbol{x}_2$ est le vecteur $\mathbf{1}$ sur $\{0,1,...,d\}$, $I = \{1,\ldots,d\}$, $k_2 = k_1 = k$ et*

$$\varepsilon = 1 - \frac{1}{2^{d+1-k}}$$

Pour faciliter la notation, nous utilisons $\boldsymbol{x}$ à la place de $\boldsymbol{x}_2$ dans la suite de la preuve. Pour $i \in \{0,\ldots,d\}$, soit $\boldsymbol{x}_{0\leftarrow i}$ l'instance dans $\{0,1\}^{d+1}$ obtenue en inversant la valeur x_i et en laissant toutes les autres valeurs de $\boldsymbol{x}$ inchangées. Soit $\boldsymbol{x}'$ l'instance obtenue à partir de $\boldsymbol{x}$ en inversant la valeur de la première entrée (c'est-à-dire $\boldsymbol{x}' = \boldsymbol{x}_{0\leftarrow 0}$). Puisque $h_2(\boldsymbol{x}) = 1$, $h_2(\boldsymbol{x}') = 1$, et $h_2(\boldsymbol{x}_{0\leftarrow i}) = 0$ pour chaque $i \in \{1,\cdots,d\}$, il s'ensuit que I est une raison suffisante pour h_2 et $\boldsymbol{x}$.

Maintenant, décomposons la fonction d'erreur sur l'enfant gauche et l'enfant droit de la racine x_0 :

$$\begin{aligned}\epsilon_{h_2,\boldsymbol{x}_2}(S) &= \frac{|\{\boldsymbol{y} \in \{0,1\}^{d+1} : h_2(\boldsymbol{y}) = h_2(\boldsymbol{x}), \boldsymbol{y}_S = \boldsymbol{x}_S\}|}{|\{\boldsymbol{y} \in \{0,1\}^{d+1} : \boldsymbol{y}_S = \boldsymbol{x}_S\}|} \\ &= \epsilon^{(l)}(S) + \epsilon^{(r)}(S)\end{aligned}$$

où

$$\begin{aligned}\epsilon^{(l)}(S) &= \frac{|\{\boldsymbol{y} \in \{0,1\}^{d+1} : h_2(\boldsymbol{y}) = h_2(\boldsymbol{x}), \boldsymbol{y}_S = \boldsymbol{x}_S, y_0 = 0\}|}{|\{\boldsymbol{y} \in \{0,1\}^{d+1} : \boldsymbol{y}_S = \boldsymbol{x}_S\}|} \\ &= \frac{|\{\boldsymbol{z} \in \{0,1\}^{d} : h_1(\boldsymbol{z}) = h_1(\boldsymbol{x}_1), \boldsymbol{z}_S = (\boldsymbol{x}_1)_S\}|}{2|\{\boldsymbol{z} \in \{0,1\}^{d} : \boldsymbol{z}_S = (\boldsymbol{x}_1)_S\}|} \\ &= \frac{\epsilon_{h_1,\boldsymbol{x}_1}(S)}{2}\end{aligned}$$

et où

$$\begin{aligned}\epsilon^{(r)}(S) &= \frac{|\{\boldsymbol{y} \in \{0,1\}^{d+1} : h_2(\boldsymbol{y}) = h_2(\boldsymbol{x}), \boldsymbol{y}_S = \boldsymbol{x}_S, y_0 = 1\}|}{|\{\boldsymbol{y} \in \{0,1\}^{d+1} : \boldsymbol{y}_S = \boldsymbol{x}_S\}|} \\ &= 1 - \frac{1}{2^{d+1-|S|}}\end{aligned}$$

Remarquons que $\epsilon^{(r)}$ décroît monotoniquement avec la taille de S. Notamment, $\epsilon_{h_2,\boldsymbol{x}_2}(S) > \varepsilon$ dès lors que $|S| < k$. Ainsi, $\epsilon_{h_2,\boldsymbol{x}_2}(S) \leq \varepsilon$ si et seulement si $|S| = k$ et $\epsilon_{h_1,\boldsymbol{x}_1}(S) = 0$. Avec ce résultat en main, si $S \subseteq I$ est une solution de $\boldsymbol{P}_2$, alors S est une solution de $\boldsymbol{P}_1$, puisque $I = [d]$. De manière duale, s'il n'y a pas de solution $S \subseteq I$ de taille k pour $\boldsymbol{P}_2$, alors il n'y a pas de solution $S' \subseteq [d]$ de taille au plus k pour $\boldsymbol{P}_1$, car $\epsilon_{h_1,\boldsymbol{x}_1}(S) > 0$ implique $\epsilon_{h_1,\boldsymbol{x}_1}(S') > 0$ dès lors que $S' \subseteq S$. Par conséquent, résoudre $\boldsymbol{P}_2$ en temps polynomial implique que $\boldsymbol{P}_1$ peut être résolu en temps polynomial, ce qui contredit le fait que $\boldsymbol{P}_1$ est NP*-difficile.*

8.3.2 Minimisation de l'erreur d'explication non normalisée

Compte tenu de l'exemple 45, nous pouvons constater que la fonction d'erreur $\epsilon_{h,\boldsymbol{x}}(\cdot)$ n'est généralement *pas* super-modulaire ou sous-modulaire, et *pas* monotone décroissante ou monotone croissante. Cependant, si nous nous concentrons plutôt sur la version non-normalisée $\mu_{h,\boldsymbol{x}}(\cdot)$ donnée dans (7), alors les propriétés suivantes peuvent être déduites.

Dans la suite, et étant $f : 2^{[d]} \to \mathbb{R}$, nous utilisons les quantités suivantes :

$$\begin{aligned}\mathrm{L}_f(i \mid S) &= f(S \setminus \{i\}) - f(S), \text{ et} \\ \mathrm{G}_f(i \mid S) &= f(S \cup \{i\}) - f(S)\end{aligned}$$

Rappelons que ces termes capturent respectivement la *perte marginale* de retirer un élément i d'un ensemble S et le *gain marginal* d'ajouter un élément i à un ensemble S. Pour une fonction d'ensemble f positive et un sous-ensemble non vide $I \subseteq [d]$, la *courbure* de f sur 2^I est donnée par

$$c = 1 - \min_{i \in I} \frac{\mathrm{L}_f(i \mid I)}{\mathrm{L}_f(i \mid \{i\})} = 1 - \min_{i \in I} \frac{\mathrm{G}_f(i \mid I \setminus \{i\})}{\mathrm{G}_f(i \mid \varnothing)} \tag{9}$$

Il est clair que $c \in [0,1]$ chaque fois que f est croissante et sous-modulaire, ou décroissante et super-modulaire. Notons que la courbure coïncide avec la notion de « *steepness* » définie dans [Il'01].

Lorsque $I = [d]$, c est appelée la *courbure totale* de f [CC84]. Dans le cas où f est décroissante et super-modulaire, rappelons que la condition $c < 1$ est suffisante pour garantir que la tâche de minimiser f sous contrainte de cardinalité est approximable à une constante près [LS17].

Proposition 27. *Soit $h : \{0,1\}^d \to \{0,1\}$ un classifieur, une instance $\boldsymbol{x} \in \{0,1\}^d$, et $I \subseteq [d]$ un ensemble non vide. $\mu_{h,\boldsymbol{x}}(\cdot)$ est super-modulaire et monotone décroissante. De plus, si I est une raison suffisante pour $\boldsymbol{x}$ étant donné h, alors la courbure c de $\mu_{h,\boldsymbol{x}}(\cdot)$ sur 2^I satisfait $c < 1$.*

Preuve 9. *Soit f une fonction $\mu_{h,\boldsymbol{x}}(\cdot)$, et N l'ensemble des instances $\boldsymbol{y} \in \{0,1\}^d$ tel que $h(\boldsymbol{x}) \neq h(\boldsymbol{y})$. Pour tout sous-ensemble $S \subseteq [d]$, soit $C(\boldsymbol{x}_S)$ l'ensemble des instances $\boldsymbol{y} \in \{0,1\}^d$ qui sont couvertes par $\boldsymbol{x}_S$, et pour chaque attribut $i \in S$, soit $\overline{C}(\boldsymbol{x}_{S\setminus\{i\}})$ qui représente l'ensemble $C(\boldsymbol{x}_{S\setminus\{i\}}) \setminus C(\boldsymbol{x}_S)$.*

*Le fait que f est croissante découle directement de l'observation que $\mathrm{L}_f(i \mid S) = |\overline{C}(\boldsymbol{x}_{S\setminus\{i\}}) \cap N| \geq 0$ pour tout $S \subseteq [d]$ et tout $i \in S$. Maintenant, étant donné n'importe quel **sur-ensemble** T de S, on a $\overline{C}(\boldsymbol{x}_{T\setminus\{i\}}) \subseteq \overline{C}(\boldsymbol{x}_{S\setminus\{i\}})$. Il en découle que $\overline{C}(\boldsymbol{x}_{T\setminus\{i\}}) \cap N \subseteq \overline{C}(\boldsymbol{x}_{S\setminus\{i\}}) \cap N$, et ainsi, $\mathrm{L}_f(i \mid T) \leq \mathrm{L}_f(i \mid S)$. Par conséquent, f est super-modulaire.*

En conclusion, $\mathrm{L}_f(i \mid I) > 0$ chaque fois que I est une raison suffisante pour $\boldsymbol{x}$ étant donné h. Donc, en conjonction avec le fait que, grâce à la super-modularité, $\mathrm{L}_f(i \mid \{i\}) \geq \mathrm{L}_f(i \mid I)$ pour tout $i \in I$, on obtient $c \in [0,1)$.

8.4 Algorithmes approchés

Après avoir fourni une vue d'ensemble des explications probabilistes et de la minimisation super-modulaire, nous allons maintenant présenter deux algorithmes gloutons pour l'approximation du problème 1. Ces algorithmes jouent un rôle essentiel dans la résolution approchée du problème spécifique que nous avons identifié précédemment. Nous détaillons chaque algorithme, expliquons comment il fonctionne et discutons de son efficacité par rapport à la méthode exacte basé sur l'encodage SAT. L'objectif de cette section est de fournir aux lecteurs une compréhension approfondie de la manière dont les concepts d'explication probabiliste et de minimisation super-modulaire sont mis en œuvre pour résoudre des problèmes concrets, tout en maintenant une approche d'approximation efficace.

8.4.1 Descente gloutonne (GD)

Une approche naturelle pour minimiser une fonction f super-modulaire et décroissante sous contrainte de cardinalité $|S| \leq k$ est de partir de l'ensemble d'entrée I des attributs candidates, et de retirer itérativement de la solution courante S tout attribut i qui minimise la perte marginale $\mathrm{L}_f(i \mid S)$, jusqu'à ce que la taille désirée $|S| = k$ soit atteinte. Comme l'a montré [Il'01], cette méthode gloutonne atteint une borne d'approximation de $\frac{e^p-1}{p}$, où $p = \frac{c}{1-c}$, et c est la courbure de f sur 2^I.

Dans le cadre de notre étude, la fonction d'erreur $\epsilon_{h,\boldsymbol{x}}(\cdot)$ définie dans (6) est une version normalisé de $\mu_{h,\boldsymbol{x}}(\cdot)$, qui est super-modulaire et décroissante. De plus, le facteur de normalisation $2^{d-|S|}$ est *constant* pour tous les sous-ensembles S de même taille. En se basant sur ces propriétés, nous pouvons combiner l'approche gloutonne décrite ci-dessus pour $f = \mu_{h,\boldsymbol{x}}(\cdot)$ avec une méthode de sélection par niveau qui stocke les ensembles $S_0, S_1, \cdots, S_k$ obtenus pour chaque niveau $j \in \{0, 1, \cdots, k\}$, et qui retourne de cette séquence le meilleur sous-ensemble S_j par rapport à $\epsilon_{h,\boldsymbol{x}}(\cdot)$. Une description formelle est donnée par l'algorithme 5.

Proposition 28. *Soit S^* une solution optimale du problème 1, soit c la courbure de $\mu_{h,\boldsymbol{x}}(\cdot)$ sur 2^I, et supposons que I soit une raison suffisante pour $\boldsymbol{x}$ étant donné h. La solution S_{GD} retournée par*

*l'algorithme descente gloutonne (*GD*) satisfait :*

$$\epsilon_{h,\boldsymbol{x}}(S_{\text{GD}}) \leq \left(\frac{e^p - 1}{p}\right) \epsilon_{h,\boldsymbol{x}}(S^*) \text{ où } p = \frac{c}{1-c} < \infty$$

Preuve 10. *Par application de la proposition 27, nous savons que* $0 \leq c < 1$*, et donc, par définition de* p*, nous avons* $p < \infty$*.*

Soit j^* *la taille de* S^* *et soit* S_{j^*} *la solution calculée par* GD *à la fin de l'étape* $j = n - j^* + 1$*. Notons que* $|S^*| = |S_{j^*}|$*. Ainsi, par application du corollaire 4 dans [Il'01], nous avons*

$$\mu_{h,\boldsymbol{x}}(S_{j^*}) \leq \left(\frac{e^p - 1}{p}\right) \mu_{h,\boldsymbol{x}}(S^*)$$

Enfin, puisque GD *retourne un minimiseur de* $\epsilon_{h,\boldsymbol{x}}(\cdot)$ *sur la séquence* $S_0, \cdots, S_{j^*}, \cdots, S_k$*, nous pouvons déduire que*

$$\begin{aligned}\epsilon_{h,\boldsymbol{x}}(S_{\text{GD}}) &\leq \epsilon_{h,\boldsymbol{x}}(S_{j^*}) = \frac{\mu_{h,\boldsymbol{x}}(S_{j^*})}{2^{d-j^*}} \\ &\leq \left(\frac{e^p - 1}{p}\right) \frac{\mu_{h,\boldsymbol{x}}(S^*)}{2^{d-j^*}} = \left(\frac{e^p - 1}{p}\right) \epsilon_{h,\boldsymbol{x}}(S^*)\end{aligned}$$

Algorithm 5 Descente gloutonne (GD)

Input: un classifieur h, une instance $\boldsymbol{x}$, un ensemble d'attributs I, un entier k
Output: un ensemble d'attributs $S_{\text{GD}} \subseteq I$ de taille au plus k

$S_n \leftarrow I, n \leftarrow |I|$

for $j = n$ ***downto*** 1 **do**
  $i^* \leftarrow \text{Argmin}_{i \in S_j} \mu_{h,\boldsymbol{x}}(S_j \setminus \{i\})$
  $S_{j-1} = S_j \setminus \{i^*\}$

$S_{\text{GD}} \leftarrow \text{Argmin}_{S \in \{S_0, S_1, \cdots, S_k\}} \epsilon_{h,\boldsymbol{x}}(S)$

return S_{GD}

8.4.2 Ascension gloutonne (GA)

Une approche alternative consiste à considérer la fonction objective $f = -\mu_{h,\boldsymbol{x}}(\cdot)$, qui est sous-modulaire et croissante. Basé sur la méthode gloutonne bien connue pour la maximisation sous-modulaire [NWF78], nous pourrions partir de $S_0 = \varnothing$ et ajouter itérativement à la solution actuelle S_{j-1} tout maximiseur $i \in I \setminus S_{j-1}$ du gain marginal $\text{G}_f(i \mid S_{j-1})$ jusqu'à ce que $|S_j| = k$. Malheureusement, une telle méthode échouerait ici car f n'est pas forcément *positive*. Cependant, comme l'ont observé les auteurs de [LS17], ce problème peut être atténué en augmentant légèrement la limite de taille k. Plus précisément, étant donné un paramètre $\gamma \in (0, 1)$, la méthode gloutonne atteint une approximation de $\frac{1}{1-\gamma}$, lorsqu'elle est autorisée à améliorer sa solution S_{j-1} jusqu'à ce que $|S_j| = k\lceil \ln\left(f(\varnothing)/\gamma f(S_{j-1})\right)\rceil$. En couplant cette idée avec la méthode de sélection par niveau suggérée ci-dessus, nous obtenons un algorithme d'ascension glouton pour minimiser $\epsilon_{h,\boldsymbol{x}}(\cdot)$, l'algorithme 6.

Algorithm 6 Ascension gloutonne (GA)

Input: un classifieur h, une instance $\boldsymbol{x}$, un ensemble d'attributs I, un entier k
Output: un ensemble d'attributs $S_{\text{GA}} \subseteq I$

$j \leftarrow 0$, $S_0 \leftarrow \varnothing$, $\gamma \leftarrow \max\{\frac{1}{e}, c\}$
// c est la courbure de $\mu_{h,\boldsymbol{x}}(\cdot)$ sur 2^I
repeat
  $i^* \leftarrow \text{Argmin}_{i \in I \setminus S_{j-1}} \mu_{h,\boldsymbol{x}}(S_{j-1} \cup \{i\})$
  $S_j = S_{j-1} \cup \{i^*\}$
until $j = k\left\lceil \ln\left(\dfrac{\mu_{h,\boldsymbol{x}}(\varnothing)}{\gamma \cdot \mu_{h,\boldsymbol{x}}(S_j)}\right)\right\rceil$;
$S_{\text{GA}} \leftarrow \text{Argmin}_{S \in \{S_0, S_1, \cdots, S_j\}} \epsilon_{h,\boldsymbol{x}}(S)$
return S_{GA}

Proposition 29. *Sous les mêmes conditions que celles données à la proposition 28, la solution S_{GA} retournée par ascension gloutonne (GA) satisfait :*

$$\epsilon_{h,\boldsymbol{x}}(S_{\text{GA}}) \leq \left(2^{k\lceil \ln \frac{2e}{1-c}\rceil - |S^*|}\right) \cdot \left(\frac{1}{1-\gamma}\right) \cdot \epsilon_{h,\boldsymbol{x}}(S^*), \text{ où } \gamma = \max\{\frac{1}{e}, c\}$$
$$\text{et } |S_{\text{GA}}| \leq k\left(1 + \left\lceil \ln \frac{\mu_{h,\boldsymbol{x}}(\varnothing)}{\mu_{h,\boldsymbol{x}}(S^*)}\right\rceil\right) \leq k\left\lceil \ln \frac{2e}{1-c}\right\rceil$$

Preuve 11. *Débutons par examiner la taille de S_{GA}. Par application du théorème 5 de [LS17], nous avons*

$$\begin{aligned} |S_{\text{GA}}| &\leq k\left(\left\lceil \ln \frac{\mu_{h,\boldsymbol{x}}(\varnothing)}{\gamma \cdot \mu_{h,\boldsymbol{x}}(S^*)}\right\rceil\right) \\ &\leq k\left(1 + \left\lceil \ln \frac{\mu_{h,\boldsymbol{x}}(\varnothing)}{\mu_{h,\boldsymbol{x}}(S^*)}\right\rceil\right) \end{aligned}$$

où la dernière inégalité découle de $\gamma \geq \frac{1}{e}$. D'autre part, comme I est une raison suffisante, nous savons que $\mu_{h,\boldsymbol{x}}(I) = 0$ et $\min_{i \in I} \mu_{h,\boldsymbol{x}}(I \setminus \{i\}) = 1$. Ainsi, la courbure de $\mu_{h,\boldsymbol{x}}(\cdot)$ sur 2^I vérifie

$$\begin{aligned} c &= 1 - \min_{i \in I} \frac{\text{L}_f(i \mid I)}{\text{L}_f(i \mid \{i\})} \\ &= 1 - \frac{1}{\mu_{h,\boldsymbol{x}}(\varnothing) - \min_{i \in I} \mu_{h,\boldsymbol{x}}(\{i\})} \end{aligned}$$

En réarrangeant cette dernière équation et en utilisant le fait que $\min_{i \in I} \mu_{h,\boldsymbol{x}}(\{i\}) \leq \frac{1}{2}\mu_{h,\boldsymbol{x}}(\varnothing)$, nous obtenons

$$\frac{1}{1-c} = \mu_{h,\boldsymbol{x}}(\varnothing) - \min_{i \in I} \mu_{h,\boldsymbol{x}}(\{i\}) \geq \frac{1}{2}\mu_{h,\boldsymbol{x}}(\varnothing)$$

En reportant ce résultat dans la borne sur la taille de S_{GA} et en utilisant le fait que $\mu_{h,\boldsymbol{x}}(S^*) \geq 1$, nous avons donc

$$|S_{\text{GA}}| \leq k\left(1 + \left\lceil \ln \frac{\mu_{h,\boldsymbol{x}}(\varnothing)}{\mu_{h,\boldsymbol{x}}(S^*)}\right\rceil\right) \leq k\left\lceil \ln \frac{2e}{1-c}\right\rceil$$

Avec ce résultat en main, concentrons-nous à présent sur $\epsilon_{h,\boldsymbol{x}}(S_{\text{GA}})$.

En utilisant le fait que S_{GA} *est un minimiseur de* $\epsilon_{h,\boldsymbol{x}}(\cdot)$ *sur* $\{S_0, \cdots, S_j\}$, *et en appliquant le théorème 5 de [LS17] sur* $\mu_{h,\boldsymbol{x}}(S_j)$, *nous avons*

$$\begin{aligned}\epsilon_{h,\boldsymbol{x}}(S_{\text{GA}}) \leq \epsilon_{h,\boldsymbol{x}}(S_j) &= \frac{\mu_{h,\boldsymbol{x}}(S_j)}{2^{d-j}} \\ &\leq \left(\frac{1}{1-\gamma}\right)\frac{\mu_{h,\boldsymbol{x}}(S^*)}{2^{d-j}}\end{aligned}$$

Ainsi, en couplant le fait que $2^{d-j} = (2^{d-|S^*|})(2^{|S^*|-j})$ *avec la borne supérieure trouvée pour* $j = |S_{\text{GA}}|$, *nous pouvons déduire que*

$$\begin{aligned}\epsilon_{h,\boldsymbol{x}}(S_{\text{GA}}) &\leq \left(\frac{1}{1-\gamma}\right)\frac{\mu_{h,\boldsymbol{x}}(S^*)}{(2^{d-|S^*|})(2^{|S^*|-j})} \\ &= \left(\frac{1}{1-\gamma}\right)2^{j-|S^*|}\epsilon_{h,\boldsymbol{x}}(S^*) \\ &\leq \left(\frac{1}{1-\gamma}\right)2^{k\lceil \ln \frac{2e}{1-c}\rceil - |S^*|}\epsilon_{h,\boldsymbol{x}}(S^*)\end{aligned}$$

Exemple 46. *Pour le classifieur* h *représenté par l'arbre de la figure 35 (noté* $\mathcal{T}$*). Soit* $\boldsymbol{x} = (1,1,1)$ *et* $I = [3] = \{1,2,3\}$, *et* $t_{\{1,2,3\}} = x_1 \wedge x_2 \wedge x_3$. *Nous cherchons a trouver un sous-ensemble de taille au plus* $k = 2$ *qui minimise l'erreur de classification de l'instance* $\boldsymbol{x}$, *étant donné l'arbre de décision* $\mathcal{T}$ *de la figure 35, associé au classifieur* h.

En suivant les étapes de l'algorithme 6 : nous avons d'abord $S_0 \leftarrow \varnothing$ *et* $\mu_{h,\boldsymbol{x}}(S_0) = 5$. *Ensuite, nous avons* $x_3 \leftarrow \underset{e\in\{x_1,x_2,x_3\}}{\text{argmin}}\ \mu_{h,\boldsymbol{x}}(\{e\})$, *alors* $S_1 \leftarrow \{x_3\}$ *et* $\mu_{h,\boldsymbol{x}}(S_1) = 2$. *Ensuite, nous avons* $x_1 \leftarrow \underset{e\in\{x_1,x_2\}}{\text{argmin}}\ \mu_{h,\boldsymbol{x}}(S_1 \cup \{e\})$, *alors* $S_2 \leftarrow \{x_3, x_1\}$ *et* $\mu_{h,\boldsymbol{x}}(S_2) = 1$.

Au final, nous avons $S_1 \leftarrow \text{Argmin}_{S\in\{S_0,S_1,S_2\}}\epsilon_{h,\boldsymbol{x}}(S)$, *donc* $S_{\text{GA}} \leftarrow \{x_3\}$ *et* $\epsilon_{h,\boldsymbol{x}}(S_{\text{GA}}) = \frac{1}{2}$. *L'algorithme 6 a capturé une raison* $\frac{1}{2}$*-probable de taille minimale.*

8.4.3 Application aux arbres de décision

Les bornes d'approximation données dans les propositions 28 et 29 sont valables pour *n'importe* quelle classe de classifieurs (booléens). Cependant, pour garantir que GD et GA sont efficaces d'un point de vue computationnel, chaque appel à l'oracle de valeur $\mu_{h,\boldsymbol{x}}(\cdot)$ doit s'exécuter en temps polynomial. Comme souligné à la section 8.2.3, c'est le cas pour les arbres de décision.

Plus précisément, si le classifieur d'entrée h de GD est représenté par un arbre de décision $\mathcal{T}$, alors, selon la proposition 25 et le fait que le nombre d'appels à l'oracle de valeur est quadratique en $n = |I|$, cela implique que GD s'exécute en temps $\mathcal{O}(n^3|\mathcal{T}|)$. Pour GA, le nombre d'appels à l'oracle de valeur est borné par $jn + n + 2$, où j est le nombre d'itérations de la boucle principale et $n + 2$ est le nombre d'appels requis pour calculer c. Ainsi, GA s'exécute en temps $\mathcal{O}(kn^2(1 + \ln 2/c)|\mathcal{T}|)$.

8.5 Expérimentations

Pour valider l'efficacité de nos algorithmes, nous avons considéré diverses instances du problème 1, où le classifieur d'entrée est décrit par un arbre de décision. Le code a été écrit en utilisant le langage Python. Toutes les expériences ont été conduites sur un ordinateur équipé d'un processeur Intel(R) Core $i9-9900$ cadencé à $3,1$ GHz et avec 64 GiB de RAM.

8.5.1 Protocole expérimental

Dans nos expériences, nous avons considéré $B = 50$ jeux de données, issus des référentiels standards *Kaggle*, *OpenML* et *UCI*. *mnist38* et *mnist49* sont des sous-ensembles de *mnist*. À l'exception de *cnae*, tous les benchmarks correspondent à des tâches de classification binaire avec un nombre d'attributs allant de 10^1 à 10^5. La tâche de classification multi-classes *cnae* a été transformée en une tâche de classification binaire en considérant la classe dominante (la plus représentée dans le benchmark) par rapport à toutes les autres classes.

Pour chaque benchmark $b \in [B]$, une tâche d'explication consiste en un tuple $(\mathcal{T}, \boldsymbol{x}, I, k)$ décrit comme suit. $\mathcal{T}$ est la représentation par un arbre de décision d'un classifieur h, qui a été appris à partir de l'ensemble d'entraînement de b. Dans nos expériences, nous avons utilisé une implémentation de l'algorithme CART de `Scikit-Learn` pour générer $\mathcal{T}$. La précision de h est mesurée sur l'ensemble de test de b. En interprétant chaque nœud interne de $\mathcal{T}$ comme un attribut booléen, l'instance $\boldsymbol{x}$ à expliquer est extraite de l'ensemble de test de b et binarisée en fonction des d attributs présents dans $\mathcal{T}$. L'ensemble I est donné par la raison directe pour $\boldsymbol{x}$ étant donné $\mathcal{T}$ [ABB+22d], c'est à dire les caractéristiques de $\boldsymbol{x}$ présentes dans le chemin unique de la racine à la feuille dans $\mathcal{T}$ qui est compatible avec l'instance $\boldsymbol{x}$. Enfin, nous avons utilisé $k = 7 \pm 2$ comme limite de taille. La performance de nos algorithmes sur un benchmark b est mesurée en tirant uniformément au hasard m instances $\boldsymbol{x}$ de l'ensemble de test de b et en calculant l'erreur moyenne de $\epsilon_{h,\boldsymbol{x}}(S)$ et de la taille moyenne $|S|$ de la sortie $S \subseteq I$. Dans nos expériences, m a été fixé à $\min\{s, 150\}$, où s est la taille de l'ensemble de test de b.

Pour comparer les performances de GD et GA avec une méthode exacte, nous avons choisi l'approche basée sur SAT présentée dans [ABROS22]. Plus précisément, un encodage SAT a été fourni pour la tâche suivante : étant donné en entrée un classifieur h représenté par un arbre de décision $\mathcal{T}$, une instance $\boldsymbol{x}$ et deux paramètres $k \leq d$ et $\varepsilon \in [0, 1)$, renvoyer en sortie « oui » s'il existe un ensemble d'attributs S satisfaisant à la fois $|S| \leq k$ et $\epsilon_{h,\boldsymbol{x}}(S) \leq \varepsilon$, et « non » sinon. Dans le cadre de notre protocole expérimental, S est un sous-ensemble de la raison directe I pour $\boldsymbol{x}$. L'encodage SAT sus-mentionné a été étendu à la version de décision du problème 1, en ajoutant la clause $\bigvee\{x_i : (x_I)_i = 1\} \vee \{\overline{x}_i : (x_I)_i = 0\}$. Pour la version originale du problème 1, une recherche binaire (dichotomique) sur l'intervalle $]0, 1]$ a été effectuée afin de trouver un ensemble S qui minimise $\epsilon_{h,\boldsymbol{x}}(\cdot)$ avec une précision de 10^{-3}, ce qui nécessite au plus 10 appels au solveur SAT. Nous avons utilisé le solveur GLUCOSE 4 via son implémentation dans `Pysat`, un temps limite (TO) de 30 minutes a été défini par instance.

Benchmark				$\epsilon_{h,\boldsymbol{x}}(S)$			$\lvert S\rvert$			Time (s)
name	acc	d	$\lvert I\rvert$	GA	GD	SAT	GA	GD	SAT	SAT
voting	94.66	16	3.05	0.07 (±0.09)	0.07 (±0.09)	0.07 (±0.09)	2.00	2.00	2.00	0.37
placement	95.38	18	3.62	0.14 (±0.20)	0.14 (±0.20)	0.14 (±0.20)	2.02	2.02	2.02	0.55
anneal2	98.52	14	4.03	0.19 (±0.11)	0.19 (±0.11)	0.19 (±0.11)	2.10	2.10	2.10	0.21
cars	98.36	21	4.33	0.16 (±0.11)	0.16 (±0.11)	0.16 (±0.11)	2.08	2.08	2.08	0.74
hepatitis	83.72	13	4.47	0.05 (±0.10)	0.05 (±0.10)	0.05 (±0.10)	3.00	3.00	3.00	0.41
autos	87.10	21	4.50	0.12 (±0.12)	0.12 (±0.12)	0.12 (±0.12)	3.53	3.53	3.53	0.72
backache	88.89	18	4.83	0.24 (±0.11)	0.24 (±0.11)	0.24 (±0.11)	2.22	2.22	2.22	0.66
cleveland2	75.82	35	4.98	0.05 (±0.08)	0.05 (±0.08)	0.05 (±0.08)	4.10	4.10	4.10	4.42
mushroom	100	17	4.99	0.28 (±0.15)	0.28 (±0.15)	0.28 (±0.15)	3.02	3.02	3.02	0.55
meta-data	87.42	44	5.09	0.08 (±0.11)	0.08 (±0.11)	0.08 (±0.11)	3.10	3.10	3.10	12.14
vehicle	96.85	23	5.19	0.21 (±0.10)	0.21 (±0.10)	0.21 (±0.10)	2.02	2.02	2.02	1.23
ionosphere	95.28	19	5.22	0.54 (±0.40)	0.54 (±0.40)	0.54 (±0.40)	1.03	1.03	1.03	0.39
kr-vs-kp	99.79	32	5.29	0.21 (±0.15)	0.21(±0.15)	0.21 (±0.15)	1.07	1.07	1.07	2.08
balance	85.64	17	5.37	0.06 (±0.08)	0.06 (±0.08)	0.06 (±0.08)	4.03	4.03	4.03	1.82
glass	78.46	31	5.38	0.26 (±0.11)	0.26 (±0.11)	0.26 (±0.11)	2.14	2.14	2.14	2.36
student perf.	91.79	30	5.41	0.26 (±0.11)	0.26 (±0.11)	0.26 (±0.11)	2.00	2.00	2.00	2.16
biomed	96.83	15	5.68	0.25 (±0.10)	0.25 (±0.10)	0.25 (±0.10)	2.32	2.32	2.32	0.45
tae	65.22	32	5.78	0.12 (±0.19)	0.13 (±0.20)	0.12 (±0.18)	3.96	3.91	3.94	3.22
primary tumor	84.31	23	6.23	0.09 (±0.09)	0.09 (±0.09)	0.09 (±0.08)	4.22	4.22	4.22	3.58
liver disorders	75.96	58	6.38	0.18 (±0.09)	0.18 (±0.08)	0.18 (±0.08)	4.00	4.00	4.00	27.33
schizophrenia	80.39	33	6.39	0.37 (±0.24)	0.37 (±0.24)	0.37 (±0.24)	1.27	1.27	1.27	4.79
tic-tac-toe	92.36	9	6.46	0.24 (±0.13)	0.24 (±0.13)	0.23 (±0.10)	2.96	2.87	2.87	1.43
australian	84.06	70	6.48	0.24 (±0.13)	0.24 (±0.14)	0.24 (±0.13)	3.56	3.55	3.55	63.49
hungarian	62.92	13	6.65	0.12 (±0.12)	0.12 (±0.12)	0.11 (±0.10)	3.58	3.56	3.56	1.68
horse colic	75.68	40	6.73	0.14 (±0.07)	0.13 (±0.07)	0.13 (±0.07)	4.03	4.06	4.06	11.56
madelon	66.41	500	7.33	0.06 (±0.09)	0.06 (±0.09)	–	6.50	6.50	–	–
haberman	67.39	53	8.18	0.73 (±0.11)	0.76 (±0.09)	0.75 (±0.10)	2.96	3.02	3.08	29.43
indian liver	64.57	84	8.21	0.10 (±0.09)	0.10 (±0.09)	0.16 (±0.12)	5.08	4.89	6.12	176.28
pima indians	75.32	97	8.30	0.15 (±0.14)	0.15 (±0.14)	0.16 (±0.12)	5.85	5.84	6.58	484.6
dexter	86.11	20000	8.32	0.06 (±0.08)	0.06 (±0.08)	–	5.95	5.96	–	–
loan eligibility	74.31	68	8.47	0.19 (±0.13)	0.18 (±0.13)	0.20 (±0.14)	5.60	5.70	6.82	42.87
patient treat.	66.01	10	8.92	0.05 (±0.09)	0.03 (±0.06)	0.03 (±0.08)	5.63	5.94	5.94	24.08
wine	69.58	11	9.03	0.09 (±0.10)	0.09 (±0.09)	0.09 (±0.12)	5.59	5.64	5.62	36.32
christine	50.8	1023	9.47	0.28 (±0.10)	0.28 (±0.10)	–	7.00	6.99	–	–
gina agnostic	85.31	970	9.69	0.20 (±0.15)	0.20 (±0.15)	–	6.90	6.90	–	–
email spam	93.88	3000	10.46	0.09 (±0.13)	0.09 (±0.13)	–	5.22	5.22	–	–
gina prior	85.59	784	10.53	0.27 (±0.16)	0.27 (±0.16)	–	6.89	6.89	–	–
employee attr.	82.45	63	10.56	0.06 (±0.09)	0.06 (±0.09)	0.20 (±0.11)	6.41	6.39	6.98	1017.24
contraceptive	51.36	90	10.84	0.06 (±0.08)	0.06 (±0.08)	0.39 (±0.17)	4.27	4.26	5.95	1096.07
compas	67.60	40	10.95	0.03 (±0.07)	0.04 (±0.08)	0.05 (±0.09)	5.68	5.83	6.78	1082.32
fetal health	91.85	93	11.33	0.12 (±0.06)	0.12 (±0.06)	0.23 (±0.11)	5.59	5.59	6.00	930.61
dorothea	91.88	100000	12.90	0.25 (±0.10)	0.25 (±0.10)	–	6.70	6.70	–	–
bank market.	89.49	882	13.11	0.29 (±0.08)	0.29 (±0.07)	–	6.99	6.99	–	–
mnist49	95.99	784	15.57	0.37 (±0.14)	0.37 (±0.14)	–	6.97	6.89	–	–
spambase	92.11	236	16.09	0.24 (±0.11)	0.23 (±0.09)	–	6.87	6.87	–	–
adult	81.16	2433	16.43	0.33 (±0.12)	0.33 (±0.12)	–	6.87	6.87	–	–
mnist38	96.42	784	17.89	0.37 (±0.13)	0.38 (±0.14)	–	6.93	6.93	–	–
cnae	92.59	856	19.07	0.32 (±0.25)	0.32 (±0.25)	–	5.97	5.97	–	–
gisette	94.10	5000	21.42	0.32 (±0.11)	0.32 (±0.11)	–	6.88	6.88	–	–
farm ads	80.78	54877	23.15	0.13 (±0.17)	0.13 (±0.17)	–	6.31	6.31	–	–

TABLE 9 – Résultats expérimentaux sur 50 benchmarks pour les explications issues d'arbres de décision, en utilisant $k = 7$

8.5.2 Résultats

Dans le tableau 9, nous présentons nos résultats sur 50 benchmarks, pour $k = 7$. La colonne la plus à gauche donne le nom de benchmark b. Les colonnes acc et d donnent respectivement la précision et le nombre d'attributs binaires de l'arbre de décision. Les lignes sont triées en fonction de la taille moyenne $|I|$ de la raison directe. Les cinquième, sixième et septième colonnes présentent l'erreur moyenne $\epsilon_{h,\boldsymbol{x}}(S)$ de l'explication S renvoyée par GA, GD, et l'approche basée sur l'encodage SAT, respectivement. Les trois colonnes suivantes rapportent la taille moyenne de S pour ces algorithmes. Enfin, la dernière colonne donne les temps d'exécution moyens (en secondes) de l'approche basée sur l'encodage SAT. Il est important de noter que, pour les benchmarks en *bleu*, le solveur SAT atteint le temps limite de 30 minutes avant même la fin de la recherche dichotomique, ce qui entraîne une dégradation de la précision. Pour les benchmarks en *magenta*, le solveur n'arrive pas à donner des résultats même pour une seule instance avant d'atteindre le temps limite de 30 minutes. Nous n'avons pas rapporté les temps d'exécution de GA et GD car ces algorithmes ont toujours pu fournir une solution en moins de 0.1 secondes pour tous les benchmarks.

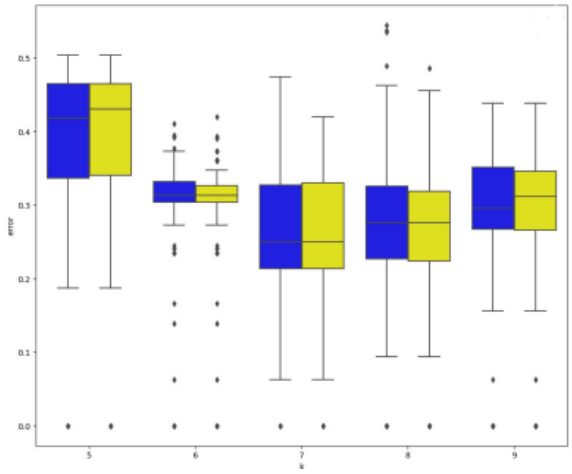
À la lumière de ces résultats, nous pouvons observer que les performances des algorithmes gloutons pour minimiser $\epsilon_{h,\boldsymbol{x}}(\cdot)$ sont remarquables, en particulier en comparaison avec les performances de l'approche exacte basée sur l'encodage SAT. Pour les benchmarks où le solveur SAT pouvait renvoyer une solution optimale S^*, les différences $\epsilon_{h,\boldsymbol{x}}(S_{\text{GD}}) - \epsilon_{h,\boldsymbol{x}}(S^*)$ et $\epsilon_{h,\boldsymbol{x}}(S_{\text{GA}}) - \epsilon_{h,\boldsymbol{x}}(S^*)$ sont le plus souvent négligeables. De plus, pour les benchmarks de grande dimension tels que *dorothea*, *gisette* et *farm ads*, GA et GD restent stables en fournissant des explications avec des erreurs comparables en quelques dixièmes de milliseconde. En ce qui concerne la concision des explications, nous pouvons voir que $|S_{\text{GD}}|$ est en moyenne plus petit que $|S^*|$. $|S_{\text{GA}}|$ est en moyenne plus petit que la taille limite de $k = 7$, ce qui indique que le majorant de $|S_{\text{GA}}|$ donné à la proposition 29 est rarement atteint en pratique. Enfin, GA et GD ont pu réduire efficacement les raisons directes I considérées. En d'autres termes, les deux algorithmes sont, en pratique, suffisamment robustes pour traiter certaines tâches d'explication pour lesquelles la courbure c de la fonction d'erreur non normalisée est proche de 1 ou égale à 1.

8.5.3 Expérimentations supplémentaires

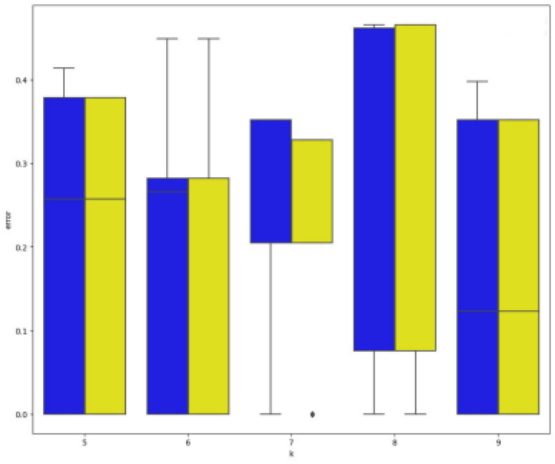
Résultats pour $k = 7 \pm 2$. Dans la figure 38, 6 diagrammes à barres fusionnés présentent les résultats d'erreur pour GA et GD lorsque k varie de 5 à 9. À noter que pour 3 de ces 4 benchmarks, le solveur SAT n'a pas pu terminer la recherche dichotomique avant d'atteindre le temps limite de 30 min. Nous pouvons observer que les performances de GA et GD sont très similaires. Une exception notable est le jeu de données *compas*, où GA est légèrement meilleur que GD lorsque k augmente.

Résultats pour $k = 7$ et $I = t_{\boldsymbol{x}}$. Comme souligné par [IIMS20], les raisons directes ne sont pas nécessairement des raisons suffisantes, car les arbres de décision peuvent inclure des caractéristiques non pertinentes. Néanmoins, les résultats ci-dessus indiquent que GA et GD sont suffisamment robustes pour réduire de telles explications d'entrée. En approfondissant l'analyse, on peut se demander si ces algorithmes d'approximation sont capables de trouver des explications probabilistes lorsque I n'est plus une raison directe pour $\boldsymbol{x}$ etant donné $\mathcal{T}$, mais comprend toutes les caractéristiques présentes dans $\boldsymbol{x}$ (c'est-à-dire $I = t_{\boldsymbol{x}}$). Les résultats correspondants sont rapportés dans le tableau 10.

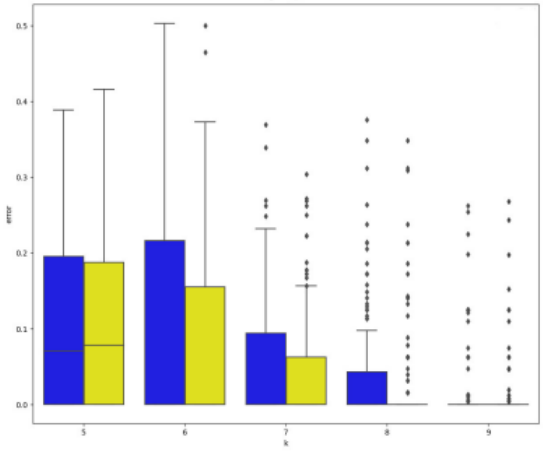
Sans surprise, les performances de l'approche basée sur SAT se dégradent lorsque d augmente. On peut également observer que les performances de GD sont légèrement moins bonnes que celles de GA. À partir de $S_n = t_{\boldsymbol{x}}$, GD doit effectuer d itérations de la boucle principale, ce qui augmente les chances d'atteindre un mauvais optimum local. En revanche, GA n'effectue que $\mathcal{O}(k \ln(1/\gamma))$ itérations de la boucle principale pour sélectionner une solution proche de l'optimum trouvé par l'approche SAT.



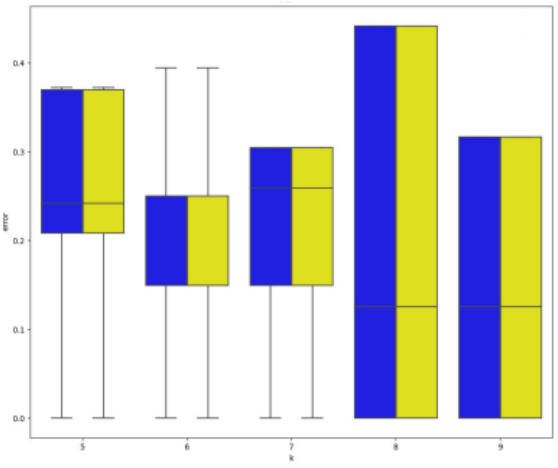
(a) *adult*



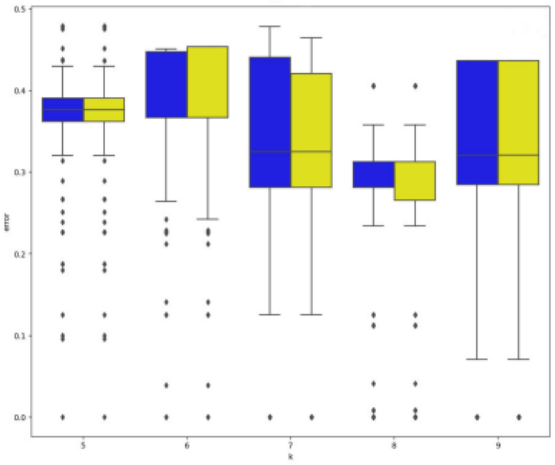
(b) *cnae*



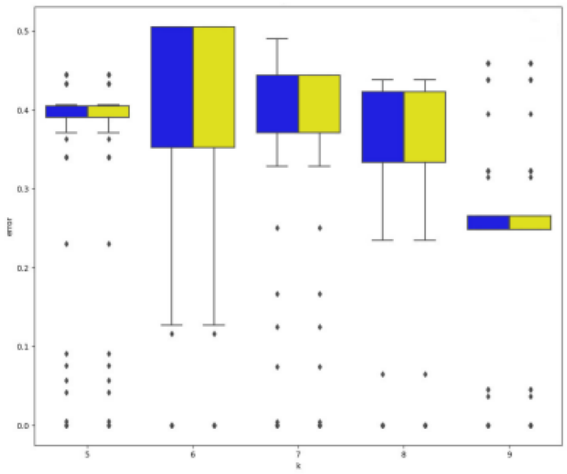
(c) *compas*



(d) *farm-ads*



(e) *mnist49*



(f) *gisette*

FIGURE 38 – Diagrammes à barres pour les erreurs de GA (jaune) et GD (bleu), en utilisant $k = 7 \pm 2$.

Benchmark			$\epsilon_{h,x}(S)$			|S|			Time (s)		
name	acc	d	GA	GD	SAT	GA	GD	SAT	GA	GD	SAT
shuttle	94.87	9	0.0019 (±0.0229)	0.0017 (±0.0203)	0.0017 (±0.0203)	2.15	2.75	4.26	0.0002	0.0004	1.02
tic-tac-toe	95.58	9	0.0008 (±0.0102)	0.0092 (±0.0326)	0.0008 (±0.0101)	4.68	4.97	5.85	0.0006	0.0008	0.79
patient treat.	66.77	10	0.0154 (±0.0555)	0.0157 (±0.0664)	0.0152 (±0.0574)	5.67	6.37	6.52	0.0023	0.0034	35.30
wine	70.0	11	0.0150 (±0.0406)	0.0308 (±0.0487)	0.0153 (±0.0383)	5.98	6.66	6.90	0.0025	0.0037	25.83
hepatitis	81.74	13	0.0038 (±0.0377)	0.0087 (±0.0318)	0.0034 (±0.0187)	2.86	4.07	3.85	0.0002	0.0002	0.63
hungarian	80.90	13	0.0075 (±0.0503)	0.0116 (±0.0271)	0.0075 (±0.0442)	5.46	5.80	6.90	0.0009	0.0013	1.53
anneal2	99.63	14	0.0248 (±0.0498)	0.0332 (±0.0450)	0.0245 (±0.0490)	3.66	3.98	5.00	0.0002	0.0003	0.32
biomed	93.65	15	0.0000 (±0.0000)	0.0000 (±0.0000)	0.0009 (±0.0000)	4.52	4.59	4.52	0.0004	0.0005	0.47
voting	93.89	16	0.0026 (±0.0188)	0.0048 (±0.0269)	0.0026 (±0.0164)	3.01	3.78	5.02	0.0003	0.0006	0.54
balance	81.38	17	0.0032 (±0.0118)	0.0121 (±0.0183)	0.0020 (±0.0084)	5.45	5.77	5.85	0.0014	0.0025	27.22
mushroom	100.0	17	0.0009 (±0.0066)	0.0052 (±0.0222)	0.0005 (±0.0015)	4.55	4.87	5.45	0.0004	0.0007	2.59
backache	85.19	18	0.0100 (±0.0357)	0.0535 (±0.0512)	0.0080 (±0.0237)	3.43	4.43	4.92	0.0003	0.0006	1.16
placement	92.31	18	0.0000 (±0.0000)	0.0010 (±0.0291)	0.0000 (±0.0000)	3.66	4.15	5.10	0.0002	0.0002	0.32
ionosphere	96.23	19	0.0121 (±0.0490)	0.0301 (±0.0493)	0.0075 (±0.0355)	4.36	5.00	5.85	0.0005	0.0001	1.51
autos	85.48	21	0.0499 (±0.0741)	0.0838 (±0.0650)	0.0045 (±0.0219)	4.48	4.90	6.12	0.0005	0.0009	5.71
cars	91.80	21	0.0329 (±0.0864)	0.0701 (±0.1120)	0.0238 (±0.0550)	3.47	3.71	5.05	0.0006	0.0013	4.70
primary tumour	81.37	23	0.0206 (±0.0493)	0.0464 (±0.0459)	0.0074 (±0.0221)	5.37	5.74	6.08	0.0013	0.0030	47.23
vehicle	96.06	23	0.0304 (±0.0458)	0.0619 (±0.0608)	0.0143 (±0.0344)	4.49	4.77	4.90	0.0008	0.0014	14.31
breast cancer	92.98	30	0.0091 (±0.0293)	0.0244 (±0.0439)	0.0066 (±0.0237)	4.43	4.94	5.52	0.0005	0.0009	4.84
student perf	90.77	30	0.0077 (±0.0316)	0.0219 (±0.0394)	0.0005 (±0.0010)	4.23	5.00	5.75	0.0007	0.0016	10.50
glass	87.69	31	0.1168 (±0.0791)	0.1410 (±0.0906)	0.0772 (±0.0758)	4.75	4.85	5.50	0.0010	0.0019	56.90
kr-vs-kp	99.79	32	0.0096 (±0.0299)	0.0300 (±0.0573)	0.0095 (±0.0285)	3.07	3.52	4.40	0.0015	0.0047	34.30
tae	69.57	32	0.0339 (±0.0634)	0.0385 (±0.0950)	0.0236 (±0.0492)	5.20	5.37	5.82	0.0011	0.0022	72.75
schizophrenia	90.20	33	0.0022 (±0.0142)	0.0024 (±0.0089)	0.0019 (±0.0070)	4.29	4.48	6.80	0.0014	0.0037	46.06
cleveland2	65.93	35	0.1217 (±0.0797)	0.1594 (±0.0923)	0.0626 (±0.0750)	4.87	4.93	5.04	0.0017	0.0048	301.59
haberman	67.39	53	0.7467 (±0.1123)	0.7941 (±0.1074)	0.7430 (±0.1252)	2.96	3.02	3.08	0.0034	0.0041	31.50
compas	66.41	40	0.0569 (±0.0697)	0.0694 (±0.0783)	0.1586 (±0.0879)	5.65	6.55	7.00	0.0298	0.0931	1486.50
horse colic	79.28	40	0.0503 (±0.0699)	0.0342 (±0.0543)	0.0133 (±0.0176)	6.43	6.70	7.00	0.0028	0.0059	671.60
meta-data	88.68	44	0.0223 (±0.0821)	0.0290 (±0.0572)	0.0199 (±0.0771)	3.62	5.54	5.96	0.0017	0.0089	171.91
employee attr.	81.88	63	0.0708 (±0.0610)	0.0960 (±0.0723)	0.2212 (±0.1525)	6.25	6.45	7.00	0.0958	0.1032	1340.60
loan eligibility	70.14	68	0.0629 (±0.0616)	0.0870 (±0.0668)	0.0956 (±0.0500)	6.10	6.00	6.50	0.0090	0.0378	856.10
australian	79.23	70	0.0000 (±0.0000)	0.0311 (±0.0592)	0.0137 (±0.0286)	4.70	6.40	6.10	0.0041	0.0308	573.94
liver disorders	71.43	84	0.0939 (±0.1087)	0.1101 (±0.0891)	0.1278 (±0.0878)	6.03	6.90	7.00	0.0150	0.0744	897.81
contraceptive	51.36	90	0.0600 (±0.0800)	0.0600 (±0.0800)	0.3910 (±0.1700)	4.27	4.26	5.95	0.0452	0.0892	1096.07
fetal health	91.85	93	0.1200 (±0.0600)	0.1200 (±0.0600)	0.2310 (±0.1100)	5.59	5.59	6.25	0.0568	0.0723	930.61
pimas indians	72.56	97	0.1181 (±0.0834)	0.1897 (±0.1072)	0.1812 (±0.0622)	6.75	7.00	6.95	0.0183	0.1675	709.57
spambase	92.03	236	0.1268 (±0.0645)	0.1845 (±0.0813)	–	6.89	6.99	–	0.0974	0.1828	–
madelon	71.03	500	0.1257 (±0.1198)	0.1971 (±0.0930)	–	6.85	6.98	–	0.0709	0.2611	–
gina-prior	87.13	784	0.2073 (±0.0837)	0.2496 (±0.0874)	–	6.97	6.99	–	0.1087	1.3443	–
mnist38	95.82	784	0.1561 (±0.0743)	0.2675 (±0.0698)	–	6.95	7.00	–	0.1082	1.4738	–
mnist49	95.02	784	0.1857 (±0.0731)	0.2867 (±0.0970)	–	6.97	6.98	–	0.2019	1.3611	–
bank market.	89.49	882	0.3245 (±0.1201)	0.3890 (±0.1267)	–	6.99	6.99	–	0.2631	1.4288	–
gina-agnostic	83.86	970	0.1593 (±0.0792)	0.2102 (±0.1084)	–	6.96	7.00	–	0.1306	1.7953	–
christine	50.37	1023	0.2710 (±0.0678)	0.3207 (±0.0816)	–	7.00	7.00	–	0.6179	1.9402	–
adult	81.16	2433	0.3882 (±0.1521)	0.3925 (±0.1877)	–	6.87	6.87	–	0.1461	0.9339	–
email spam	94.07	3000	0.1090 (±0.1494)	0.1761 (±0.2169)	–	4.90	6.78	–	0.0704	1.0079	–
gisette	93.81	5000	0.1309 (±0.0706)	0.2006 (±0.1056)	–	6.87	6.94	–	0.0746	2.8047	–
dexter	83.89	20000	0.0321 (±0.0532)	0.0614 (±0.0525)	–	5.74	6.63	–	0.1250	0.2330	–
farm ads	80.78	54877	0.1300 (±0.1700)	0.1300 (±0.1700)	–	6.31	6.31	–	0.2710	0.9892	–
dorothea	93.04	10^5	0.2383 (±0.0743)	0.2514 (±0.0758)	–	6.87	6.97	–	0.2027	1.2040	–

TABLE 10 – Résultats expérimentaux sur 50 benchmarks, en utilisant $I = t_{\boldsymbol{x}}$ et $k = 7$.

8.6 Discussion

Dans notre étude, les explications probabilistes ont été examinées à travers le prisme de la minimisation super-modulaire. Inspirés des résultats de [Il'01, LS17], nous avons proposé deux algorithmes d'approximation gloutons pour minimiser les erreurs d'explication sous une contrainte de cardinalité, dont les performances dépendent essentiellement de la courbure c de la fonction d'erreur non normalisée $\mu_{h,\boldsymbol{x}}(\cdot)$. Il est important de noter que nos résultats d'approximation s'appliquent à n'importe quelle classe de classifieurs (booléens), et donc, nos algorithmes gloutons sont efficaces en termes de calcul chaque fois que $\mu_{h,\boldsymbol{x}}(\cdot)$ peut être évalué en temps polynomial. Au-delà des arbres de décision, qui ont été examinés dans ce chapitre, les diagrammes de décision binaires [HHS22] et les circuits dDNNF [HHS22] sont des exemples de classifieurs satisfaisant cette condition.

Des approches heuristiques pour les explications probabilistes ont été envisagées dans [IHI+22]. La tâche d'optimisation est symétrique de celle du problème 1 : étant donné un classifieur h, une instance $\boldsymbol{x}$, un ensemble d'attributs I et un paramètre d'erreur ε, l'objectif est de trouver une explication $(1-\varepsilon)$-probable $S \subseteq I$ qui minimise $|S|$. Pour cette tâche, les auteurs ont proposé un algorithme glouton qui s'exécute en temps polynomial, lorsque h est représenté par un arbre de décision $\mathcal{T}$, et I est une raison suffisante pour $\boldsymbol{x}$ et $\mathcal{T}$. Cependant, cet algorithme ne fournit aucune garantie d'approximation par rapport à la taille optimale.

À notre connaissance, les approches d'approximation pour les explications probabilistes n'ont été étudiées que dans [BLT21]. Encore une fois, le problème en question consiste à trouver une explication $(1-\varepsilon)$-probable S qui minimise $|S|$. Basé sur certains résultats sur l'apprentissage implicite, les auteurs ont proposé un algorithme polynomial de style PAC qui prend en entrée un classifieur h, une instance $\boldsymbol{x}$, un paramètre de confiance δ, et un paramètre de précision ε, et qui renvoie en sortie un ensemble $S \subseteq [d]$ avec les garanties suivantes : (i) $|S|$ est polynomial en d, $1/\delta$ et $1/\varepsilon$, et (ii) si $\boldsymbol{x}$ est tirée uniformément au hasard dans $\{0,1\}^d$, alors $\epsilon_{h,\boldsymbol{x}} \leq \varepsilon$ avec une probabilité d'au moins $(1-\delta)$. Cependant, cet algorithme est principalement d'intérêt théorique, car $|S|$ est en $\mathcal{O}\left((1/\delta)^9 (1/\epsilon)^{12}\right)$ et, plus important encore, les instances à expliquer dans les applications pratiques sont rarement choisies au hasard selon la distribution uniforme.

8.7 Conclusion

De nombreux travaux sur l'étude de la complexité des explications abductives de taille minimale ont montré que calculer une explication abductive de taille minimale est très difficile, même pour les classifieurs les plus intelligibles, tels que les arbres de décision. Des encodages SAT ont été proposés pour la recherche de telles explications [ABB+22d, IIMS20, ABB+22a].

D'autres travaux ont étudié la complexité du calcul d'une explication probabiliste [WMHK21] étant donnée une formule CNF, et plus particulièrement pour les arbres de décision [ABROS22]. Ces travaux ont montré que le problème du calcul d'une explication est hors de portée en général et qu'il est NP-difficile pour les arbres de décision. Des méthodes exactes basées sur des encodages SAT et l'utilisation de solveurs SAT modernes ont été proposées [ABROS22, IHI+22] pour le calcul exact d'une explication probabiliste. Différentes expérimentations ont montré que le calcul d'une explication probabiliste ou d'une explication abductive de taille minimale est hors de portée dans de nombreux cas pratiques. De notre côté, nous avons proposé des méthodes pour approcher efficacement les explications probabilistes via l'optimisation super-modulaire, tout en offrant des garanties mathématiques.

Conclusion de la partie « contributions »

Dans cette partie, nous avons présenté les contributions obtenues durant la préparation de cette thèse.

Une première contribution a consisté en une étude de *l'intelligibilité computationnelle des classifieurs booléens*, caractérisée par leur capacité à répondre aux requêtes d'explicabilité en temps polynomial. Les classifieurs pris en considération comprennent les arbres de décision, les formules DNF, les listes de décision, les règles de décision, les arbres boostés, et les réseaux de neurones binaires. En utilisant 9 requêtes XAI, comprenant à la fois des requêtes d'explication et des requêtes de vérification, nous montrons l'existence d'un écart significatif d'intelligibilité entre les familles de classifieurs. D'une part, les 9 requêtes XAI sont toutes traitables pour les arbres de décision. D'autre part, sauf si $\mathsf{P} \neq \mathsf{NP}$, aucune d'entre elles n'est traitable pour les formules DNF, les listes de décision, les forêts aléatoires, les arbres de décision boostés, les perceptrons multi-couches booléens, ou les réseaux de neurones binaires.

A la lumière des résultats obtenus, nous nous sommes focalisés sur les arbres de décision. Notre deuxième contribution a concerné le *pouvoir explicatif* des *arbres de décision*. Nous avons examiné la capacité des arbres de décision binaires à extraire, minimiser et compter des explications abductives / contrastives. Nous avons montré que l'ensemble de toutes les explications abductives irredondantes (ou raisons suffisantes) d'une instance peut être de taille exponentielle. Aussi, générer l'intégralité de cet ensemble peut se révéler hors de portée. De plus, deux raisons suffisantes d'une même instance peuvent différer sur toutes leurs caractéristiques. Ainsi, le calcul d'une seule raison suffisante ne donne qu'une vision parcellaire des explications abductives possibles. Nous avons également introduit les notions d'attribut nécessaire / pertinent et la notion du pouvoir explicatif d'un attribut et nous avons montré que ces notions peuvent être utiles pour dériver une vue synthétique des raisons suffisantes d'une instance.

En raison de limitations cognitives, les explications abductives sont souvent de trop grande taille pour être interprétables par les humains. Dans notre troisième contribution, nous avons abordé le défi de réduire la taille de ces explications tout en maintenant une probabilité élevée de prédiction exacte. Ce problème est difficile en général, même pour les arbres de décision. Pour surmonter cette difficulté, nous avons exploré *l'approximation des explications probabilistes* en utilisant le concept de super-modularité. Nous avons développé deux algorithmes gloutons (GA et GD) pour la minimisation super-modulaire. L'efficacité de ces algorithmes dépend de la courbure de la fonction d'erreur non normalisée qui mesure la précision de l'explication. Les expériences réalisées avec les arbres de décision ont clairement démontré que nos algorithmes gloutons offrent une alternative efficace à une méthode exacte basée sur un encodage SAT.

Conclusion générale

Les travaux réalisés durant cette thèse apportent différentes contributions dans le domaine de l'intelligence artificielle explicable (XAI).

Après avoir posé les bases essentielles à la compréhension de ce manuscrit dans la première partie, la deuxième partie a présenté diverses méthodes de génération d'explications formelles. Ces approches reposent principalement sur l'utilisation d'encodages SAT et de méthodes d'optimisation combinatoire.

— Nous avons d'abord montré que l'intelligibilité computationnelle des arbres de décision est supérieure à celle de nombreux modèles d'apprentissage automatique, ce qui nous a conduit à focaliser nos recherches sur les arbres de décision.

— Nous avons ensuite proposé des algorithmes pour l'énumération de toutes les explications abductives (à l'aide des solveurs MaxSAT) et contrastives pour une instance $\boldsymbol{x}$ étant donné un arbre de décision. Nous avons constaté que l'ensemble complet de toutes les explications abductives non redondantes d'une instance peut être de taille exponentielle. De plus, les raisons suffisantes pour une même instance peuvent différer sur tous leurs attributs, ce qui implique que le calcul d'une seule raison suffisante n'est pas informatif pour un utilisateur. Pour résoudre ce problème et tenter de synthétiser l'ensemble des raisons suffisantes, nous avons proposé des algorithmes en temps polynomial pour calculer les caractéristiques de l'instance donnée qui appartiennent à chaque (resp. aucune) raison suffisante pour cette instance étant donné un arbre de décision. Nous avons aussi développé un algorithme de calcul du *pouvoir explicatif* d'un attribut d'une instance $\boldsymbol{x}$ pour un arbre de décision donné, c'est-à-dire la proportion des raisons suffisantes qui le contiennent. Quoi qu'il n'opère pas en temps polynomial, cet algorithme nous a permis en pratique de calculer dans bien des cas le pouvoir explicatif des attributs à l'aide du compteur de modèles `D4` [LM17].

— Les travaux réalisés pour synthétiser l'ensemble des raisons suffisantes nous ont également conduits à proposer des modèles de préférence et à développer des algorithmes pour générer des explications préférées exploitant les préférences de l'utilisateur. Cette démarche présente plusieurs avantages. En prenant en compte ces préférences, nous pouvons rechercher des explications abductives qui conviennent à l'utilisateur tout en réduisant considérablement leur nombre, comme nos expérimentations l'ont confirmé.

— Les performances intéressantes obtenues par les solveurs pour le calcul des explications abductives de taille minimale ne suffisent pas à garantir la compréhension de ces explications par l'utilisateur. En raison des limitations cognitives des utilisateurs humains, les explications abductives peuvent être trop grandes pour être interprétables. Dans de tels cas, calculer des explications plus courtes avec une forte probabilité peut être plus adapté. Nous avons montré que calculer de telles explications probabilistes est NP-difficile, même pour les arbres de décision. Pour faire face à la complexité élevée du calcul d'explications probabilistes, nous nous sommes tournés vers

l'approximation des explications probabilistes via le prisme de la super-modularité. Nous avons proposé deux algorithmes gloutons (GA et GD) pour la minimisation super-modulaire de la fonction d'erreur non normalisée, dont les garanties d'approximation dépendent de la courbure. Nos expérimentations ont montré que nos algorithmes gloutons sont efficaces en pratique. Pour la quasi-totalité des benchmarks considérés dans notre étude, nos deux algorithmes ont permis de calculer une solution optimale.

Perspectives de la thèse

Ces différentes contributions ouvrent de nombreuses perspectives. Tout d'abord, nous prévoyons d'approfondir notre étude sur le pouvoir explicatif des attributs tout en comparant nos résultats avec les méthodes de l'XAI de l'état de l'art, telles que SHAP et LIME. Nos premiers travaux dans cette direction ont déjà montré l'existence d'une corrélation significative entre le pouvoir explicatif d'un attribut et la valeur SHAP de cet attribut. Cette corrélation pourrait offrir des informations précieuses sur la manière dont nos explications abductives se comparent aux méthodes existantes en termes d'interprétabilité et d'efficacité. Ensuite, nous envisageons d'étendre le concept de pouvoir explicatif à des classifieurs plus complexes, tels que les forêts aléatoires et les arbres boostés. Enfin, nous envisageons également d'explorer des approches de post-traitement pour améliorer la qualité de l'ensemble d'explications abductives produites. Nous visons à développer des techniques pour identifier et éliminer les explications peu informatives et à en reformuler d'autres, ce qui contribuerait à rendre nos explications plus claires pour les utilisateurs finaux.

Une autre question cruciale à explorer concerne la robustesse des explications formelles. Une approche que nous avons commencé à développer consiste à examiner la relation entre les performances de classification de deux classifieurs booléens qui ont été appris sur le même ensemble de données, et les explications produites à partir de ces deux classifieurs. Plus précisément, nous souhaitons analyser la corrélation entre la différence symétrique des classes apprises par ces deux classifieurs, représentée par deux fonctions booléennes, et la différence symétrique des ensembles d'explications générées (à la fois contrastives et abductives), ainsi que la distance entre ces ensembles d'explications. Cette étude vise à évaluer dans quelle mesure les explications restent cohérentes et informatives lorsque deux classifieurs similaires, mais pas nécessairement identiques, sont appliqués au même benchmark. Nous nous efforcerons de quantifier cette relation et d'explorer comment les différences entre les classifieurs se traduisent par des variations dans les explications générées. Ce faisant, nous espérons mieux comprendre la stabilité des explications formelles dans des scénarios où des variations mineures concernant les classifieurs pourraient survenir.

Le travail sur l'approximation des explications probabilistes laisse plusieurs questions en suspens. Notamment, (i) quel est le facteur d'approximation optimal pour minimiser l'erreur de la classification sous une contrainte de cardinalité ? Une réponse partielle pourrait venir de [LS17], qui ont donné un algorithme presque optimal pour minimiser une fonction super-modulaire décroissante sous une contrainte de matroïde. Cependant, cette méthode est principalement d'intérêt théorique, car sa complexité computationnelle est prohibitivement élevée. Donc, (ii) peut-on trouver des algorithmes d'approximation alternatifs et presque optimaux qui sont efficaces du point de vue du calcul ? Enfin, (iii) en utilisant des méthodes d'échantillonnage, peut-on étendre les algorithmes d'approximation à des classes de classifieurs pour lesquelles le problème d'évaluation de $\mu_{h,\boldsymbol{x}}(\cdot)$ est inabordable ?

En relation avec ces perspectives de recherche, nous souhaitons étendre notre étude aux classifieurs pour lesquels l'évaluation de la fonction d'erreur $\epsilon_{h,x}(\cdot)$ peut se faire en temps polynomial, et observer le

comportement de nos algorithmes gloutons sur ces derniers. Ensuite, nous prévoyons d'explorer le comportement de nos approches sur d'autres familles de classifieurs où l'évaluation de la fonction d'erreur $\epsilon_{h,x}(\cdot)$ ne peut pas se faire en temps polynomial, tout en essayant de maintenir des bornes d'approximation correctes. Une autre question intéressante est de déterminer le facteur d'approximation optimal pour minimiser l'erreur des raisons probabilistes sous une contrainte de cardinalité. Bien que certains progrès aient été réalisés dans la réponse à cette première question, notamment par [LS17], qui ont proposé un algorithme presque optimal pour minimiser une fonction super-modulaire décroissante sous une contrainte de matroïde, il est important de noter que cette méthode est principalement d'intérêt théorique en raison de sa complexité computationnelle prohibitivement élevée. Par conséquent, une autre question cruciale est de rechercher d'autres algorithmes gloutons avec des garanties mathématiques pour cette tâche complexe.

Enfin, il convient de noter que dans le contexte de l'approximation des explications probabilistes, nous envisageons tout d'abord d'étendre notre étude à des classifieurs plus complexes, notamment les réseaux de neurones, et d'explorer des méthodes basées sur l'échantillonnage. Ensuite, nous pensons pouvoir exploiter la super-modularité pour proposer des approximations efficaces du calcul des explications abductives de taille minimale pour les forêts aléatoires, plus précisément l'approximation des raisons majoritaires minimales. Nous pensons également pouvoir exploiter la super-modularité pour approcher l'ensemble d'attributs permettant de minimiser l'erreur moyenne du vote majoritaire pour les forêts aléatoires tout en respectant une contrainte de cardinalité.

Bibliographie

[AB18] A. ADADI et M. BERRADA. « Peeking Inside the Black-Box : A Survey on Explainable Artificial Intelligence (XAI) ». *IEEE Access*, 6 :52138–52160, 2018. 4.1.1

[ABB+21] Gilles AUDEMARD, Steve BELLART, Louenas BOUNIA, Frédéric KORICHE, Jean-Marie LAGNIEZ, et Pierre MARQUIS. « On the Computational Intelligibility of Boolean Classifiers ». Dans *Proc. of KR'21*, pages 74–86, 2021. II, 5, 5.2.1, 5.2.1, 5.2.1, 5.2.1, 5.2.1, 5.2.1, 5.2.2

[ABB+22a] G. AUDEMARD, S. BELLART, L. BOUNIA, F. KORICHE, J-M. LAGNIEZ, et P. MARQUIS. « Trading Complexity for Sparsity in Random Forest Explanations ». Dans *Proc. of AAAI'22*, 2022. 4.3, 8.7

[ABB+22b] G. AUDEMARD, S. BELLART, L. BOUNIA, F. KORICHE, J.M. LAGNIEZ, et P. MARQUIS. « On Preferred Abductive Explanations for Decision Trees and Random Forests ». Dans *Proc. of IJCAI'22*, 2022. 7, 7.1.1, 7.1.1, 7.1.2

[ABB+22c] Gilles AUDEMARD, Steve BELLART, Louenas BOUNIA, Frédéric KORICHE, Jean-Marie LAGNIEZ, et Pierre MARQUIS. « Sur le pouvoir explicatif des arbres de décision ». *EGC'2022*, 38, 2022. 6

[ABB+22d] Gilles AUDEMARD, Steve BELLART, Louenas BOUNIA, Frédéric KORICHE, Jean-Marie LAGNIEZ, et Pierre MARQUIS. « On the explanatory power of Boolean decision trees ». *Data Knowledge Engineering*, 142 :102088, 2022. 3.3.3, 3.3.3, 6, 6.2, 18, 20, 7.1.1, 7.1.1, 8.5.1, 8.7

[ABROS22] Marcelo ARENAS, Pablo BARCELÓ, Miguel ROMERO ORTH, et Bernardo SUBERCASEAUX. « On computing probabilistic explanations for decision trees ». *Advances in Neural Information Processing Systems*, 35 :28695–28707, 2022. 8.1, 8.5.1, 8.7

[AKM20] G. AUDEMARD, F. KORICHE, et P. MARQUIS. « On Tractable XAI Queries based on Compiled Representations ». Dans *Proc. of KR'20*, pages 838–849, 2020. 4.3.3, 5.1.1, 5.1.2, 5.2.1, 12, 14, 1, 1, 5.5, 6.1.3

[Amg23] Leila AMGOUD. « Explaining black-box classifiers : Properties and functions ». *Int. J. Approx. Reason.*, 155 :40–65, 2023. 4.3

[Ant01] Martin ANTHONY. « Discrete mathematics of neural networks : selected topics ». 01 2001. 3.1.3, 5.1.4

[BFOS84] L. BREIMAN, J. H. FRIEDMAN, R. A. OLSHEN, et C. J. STONE. *Classification and Regression Trees*. Wadsworth, 1984. 6

[BHMSV84] Robert K. BRAYTON, Gary D. HACHTEL, Curtis T. MCMULLEN, et Alberto L. SANGIOVANNI-VINCENTELLI. « Logic Minimization Algorithms for VLSI Synthesis ». Dans *The Kluwer International Series in Engineering and Computer Science*, 1984. 3

[BK23a] Louenas BOUNIA et Frederic KORICHE. « Approximating Probabilistic Explanations via Supermodular Minimization (Corrected Version) ». Dans *Uncertainty in Artificial Intelligence (UAI 2023).*, volume 216, pages 216–225, 2023. 8

[BK23b] Louenas BOUNIA et Frederic M KORICHE. « Approximating Probabilistic Explanations via Supermodular Minimization ». Dans *The 39th Conference on Uncertainty in Artificial Intelligence*, 2023. 8

[BLT21] Guy BLANC, Jane LANGE, et Li-Yang TAN. « Provably efficient, succinct, and precise explanations ». Dans A. BEYGELZIMER, Y. DAUPHIN, P. LIANG, et J. Wortman VAUGHAN, éditeurs, *Advances in Neural Information Processing Systems*, 2021. 8.6

[BMPS20] Pablo BARCELÓ, Mikaël MONET, Jorge PÉREZ, et Bernardo SUBERCASEAUX. « Model interpretability through the lens of computational complexity ». *Advances in neural information processing systems*, 33 :15487–15498, 2020. 18, 8

[Bre89] G. BREWKA. « Preferred subtheories : an extended logical framework for default reasoning ». Dans *Proc. of IJCAI'89*, pages 1043–1048, 1989. 15

[Bre96] L. BREIMAN. « Bagging predictors ». *Machine Learning*, 24 :123–140, 1996. 3.3

[Bre01] L. BREIMAN. « Random Forests ». *Machine Learning*, 45(1) :5–32, 2001. 3.1.3, 3.3, 3.3.1, 6

[BTT+18] R. BUNEL, I. TURKASLAN, Ph. H. S. TORR, P. KOHLI, et P. Kumar MUDIGONDA. « A Unified View of Piecewise Linear Neural Network Verification ». Dans *Proc. of NeurIPS'18*, pages 4795–4804, 2018. 5.1.1

[CABP14] Andrew CLARK, Basel ALOMAIR, Linda BUSHNELL, et Radha POOVENDRAN. « Minimizing Convergence Error in Multi-Agent Systems Via Leader Selection : A Supermodular Optimization Approach ». *IEEE Transactions on Automatic Control*, 59(6) :1480–1494, 2014. 2.3.3

[CBP14] Andrew CLARK, Linda BUSHNELL, et Radha POOVENDRAN. « A Supermodular Optimization Framework for Leader Selection Under Link Noise in Linear Multi-Agent Systems ». *IEEE Transactions on Automatic Control*, 59(2) :283–296, 2014. 1, 2.3.3, 2.3.3

[CC84] Michele CONFORTI et Gérard CORNUÉJOLS. « Submodular set functions, matroids and the greedy algorithm : Tight worst-case bounds and some generalizations of the Rado-Edmonds theorem ». *Discret. Appl. Math.*, 7 :251–274, 1984. 2.3.4, 8.3.2

[CCMS23] Clément CARBONNEL, Martin C. COOPER, et Joao MARQUES-SILVA. « Tractable Explaining of Multivariate Decision Trees ». Dans *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning*, pages 127–135, 8 2023. 5.4

[CG16] Tianqi CHEN et Carlos GUESTRIN. « XGBoost : A Scalable Tree Boosting System ». Dans *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 785–794, New York, NY, USA, 2016. 3.1.3, 3.3, 3.3.2, 3.3.2

[CGJ96] Ed COFFMAN, M.R. GAREY, et David JOHNSON. « *Approximation Algorithms for NP-Hard Problems* », pages 46–93. 01 1996. 1.2

[CH11] Yves CRAMA et Peter L. HAMMER. « Boolean Functions - Theory, Algorithms, and Applications ». Dans *Encyclopedia of mathematics and its applications*, 2011. 1.1.3, 1

[CKS01] Nadia CREIGNOU, Sanjeev KHANNA, et Madhu SUDAN. *Complexity Classifications of Boolean Constraint Satisfaction Problems*. Society for Industrial and Applied Mathematics, 2001. 1.2

[CMS23] Martin C. COOPER et João MARQUES-SILVA. « Tractability of explaining classifier decisions ». *Artificial Intelligence*, 2023. 5.4

[Coe19] Mark COECKELBERGH. « Artificial Intelligence, Responsibility Attribution, and a Relational Justification of Explainability ». *Science and Engineering Ethics*, 26 :2051 – 2068, 2019. 1.2

[COO71] Stephen A. COOK. « The complexity of theorem-proving procedures ». *Third annual ACM symposium on Theory of computing*, 267 :151–158, 1971. 1.2.3, 1.3.2, 2.3

[CP18] Zachary B. CHARLES et Dimitris PAPAILIOPOULOS. « Stability and Generalization of Learning Algorithms that Converge to Global Optima ». Dans *International Conference on Machine Learning*, 2018. 5.1.1

[CSHB17] Aditya CHATTOPADHYAY, Anirban SARKAR, Prantik HOWLADER, et Vineeth N. BALASUBRAMANIAN. « Grad-CAM++ : Generalized Gradient-Based Visual Explanations for Deep Convolutional Networks ». *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 839–847, 2017. 4.1.1, 4.1.2

[CZS+19] Hongge CHEN, Huan ZHANG, Si SI, Yang LI, Duane S. BONING, et Cho-Jui HSIEH. « Robustness Verification of Tree-based Models ». *ArXiv*, abs/1906.03849, 2019. 5.1.1

[CZZvdS20] Jonathan CRABBE, Yao ZHANG, William ZAME, et Mihaela van der SCHAAR. « Learning outside the Black-Box : The pursuit of interpretable models ». Dans H. LAROCHELLE, M. RANZATO, R. HADSELL, M.F. BALCAN, et H. LIN, éditeurs, *Advances in Neural Information Processing Systems*, volume 33, pages 17838–17849. Curran Associates, Inc., 2020. 5.1.1

[Dar99] Adnan DARWICHE. « Compiling Knowledge into Decomposable Negation Normal Form ». Dans *International Joint Conference on Artificial Intelligence*, 1999. 8.2.3

[Dar01] A. DARWICHE. « Decomposable negation normal form ». *Journal of the Association for Computing Machinery*, 48(4) :608–647, 2001. 6.2.2, 4

[DCLT19] Jacob DEVLIN, Ming-Wei CHANG, Kenton LEE, et Kristina TOUTANOVA. « BERT : Pre-training of Deep Bidirectional Transformers for Language Understanding ». *ArXiv*, abs/1810.04805, 2019. 3.1.3

[dCM22] Alexis de COLNET et Pierre MARQUIS. « On the Complexity of Enumerating Prime Implicants from Decision-DNNF Circuits ». *ArXiv*, abs/2301.13328, 2022. 6.2

[DH20] A. DARWICHE et A. HIRTH. « On the Reasons Behind Decisions ». Dans *Proc. of ECAI'20*, 2020. 4.3, 5.2.2, 6, 6.1.2, 6.2, 60, 6, 7.1.1

[Die00] Thomas G. DIETTERICH. « Ensemble Methods in Machine Learning ». pages 1–15, 2000. 3.3

[DK17] F. DOSHI-VELEZ et B. KIM. « A Roadmap for a Rigorous Science of Interpretability ». *CoRR*, abs/1702.08608, 2017. 3.1.3, 4.1.1, 6.5, 7.1.1, 7.4

[DM02] A. DARWICHE et P. MARQUIS. « A knowledge compilation map ». *Journal of Artificial Intelligence Research*, 17 :229–264, 2002. 9, 1, 6.2.2

[DM21] Adnan DARWICHE et Pierre MARQUIS. « On Quantifying Literals in Boolean Logic and Its Applications to Explainable AI ». *J. Artif. Intell. Res.*, 72 :285–328, 2021. 6.2, 5

[DQA+20] Marina DANILEVSKY, Kun QIAN, Ranit AHARONOV, Yannis KATSIS, Ban KAWAS, et Prithviraj SEN. « A Survey of the State of Explainable AI for Natural Language Processing ». Dans *AACL*, 2020. 4.1.2

[DSZ16] Anupam DATTA, Shayak SEN, et Yair ZICK. « Algorithmic Transparency via Quantitative Input Influence : Theory and Experiments with Learning Systems ». *2016 IEEE Symposium on Security and Privacy (SP)*, pages 598–617, 2016. 4.2.2

[EG95] Th. EITER et G. GOTTLOB. « The complexity of logic-based abduction ». *Journal of the Association for Computing Machinery*, 42(1) :3–42, 1995. 6.2.2

[eJYGI91] Alan TURING et JEAN-YVES G IRARD. « *La machine de Turing* ». 1991. 1.2

[FH17] Nicholas FROSST et Geoffrey E. HINTON. « Distilling a Neural Network Into a Soft Decision Tree ». *ArXiv*, abs/1711.09784, 2017. 3.1.3

[FHT00] Jerome FRIEDMAN, Trevor HASTIE, et Robert TIBSHIRANI. « Additive Logistic Regression : A Statistical View of Boosting ». *The Annals of Statistics*, 28 :337–407, 04 2000. 3.3

[FKT15] Dean FOSTER, Howard KARLOFF, et Justin THALER. « Variable Selection is Hard ». Dans Peter GRÜNWALD, Elad HAZAN, et Satyen KALE, éditeurs, *Proceedings of The 28th Conference on Learning Theory*, volume 40 de *Proceedings of Machine Learning Research*, pages 696–709, Paris, France, 03–06 Jul 2015. PMLR. 2.3.3

[Fri00] Jerome FRIEDMAN. « Greedy Function Approximation : A Gradient Boosting Machine ». *The Annals of Statistics*, 29, 11 2000. 3.3

[FS95] Yoav FREUND et Robert E. SCHAPIRE. « A decision-theoretic generalization of on-line learning and an application to boosting ». Dans Paul M. B. VITÁNYI, éditeur, *Computational Learning Theory, Second European Conference, EuroCOLT '95, Barcelona, Spain, March 13-15, 1995, Proceedings*, volume 904 de *Lecture Notes in Computer Science*, pages 23–37. Springer, 1995. 42

[Gar72] Graham R.-et Ullman J. D. GAREY, R. « An analysis of some packing algorithms. In Combinatorial Algorithms ». 39–47, 1972. 2.3

[GF17] B. GOODMAN et S. R. FLAXMAN. « European Union Regulations on Algorithmic Decision-Making and a "Right to Explanation" ». *AI Magazine*, 38(3) :50–57, 2017. 5.1.1

[GG22] Jonas GEIPING et Tom GOLDSTEIN. « Cramming : Training a Language Model on a Single GPU in One Day ». *ArXiv*, abs/2212.14034, 2022. 3.1.3

[GMR+19] R. GUIDOTTI, A. MONREALE, S. RUGGIERI, F. TURINI, F. GIANNOTTI, et D. PEDRESCHI. « A Survey of Methods for Explaining Black Box Models ». *ACM Computing Surveys*, 51(5) :93 :1–93 :42, 2019. 4.1.1

[GPAM+14] Ian GOODFELLOW, Jean POUGET-ABADIE, Mehdi MIRZA, Bing XU, David WARDE-FARLEY, Sherjil OZAIR, Aaron COURVILLE, et Yoshua BENGIO. « Generative Adversarial Nets ». Dans Z. GHAHRAMANI, M. WELLING, C. CORTES, N. LAWRENCE, et K.Q. WEINBERGER, éditeurs, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. 3.1.3

[Gra66] R. GRAHAM. « Bounds for Certain Multiprocessing Anomalies ». *Siam Journal on Applied Mathematics - SIAMAM*, 45, 11 1966. 2.3

[GSTT99] Boris GOLDENGORIN, Gerard SIERKSMA, Gert A. TIJSSEN, et Michael TSO. « The Data Correcting Algorithm for the Minimization of Supermodular Functions ». *Management Science*, 45 :1539–1551, 1999. 2.3.2

[Hås01] Johan HÅSTAD. « Some optimal inapproximability results ». *Electron. Colloquium Comput. Complex.*, TR97, 2001. 22

[Hau92] David HAUSSLER. « Decision Theoretic Generalizations of the PAC Model for Neural Net and Other Learning Applications ». *Inf. Comput.*, 100 :78–150, 1992. 5.1.1

[HCS+16] Itay HUBARA, Matthieu COURBARIAUX, Daniel SOUDRY, Ran EL-YANIV, et Yoshua BENGIO. « Binarized Neural Networks ». Dans D. LEE, M. SUGIYAMA, U. LUXBURG, I. GUYON, et R. GARNETT, éditeurs, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. 3.1.3, 5.1.4

[HHS22] Hao HU, Marie-José HUGUET, et M. SIALA. « Optimizing Binary Decision Diagrams with MaxSAT for classification ». Dans *AAAI Conference on Artificial Intelligence*, 2022. 8.6

[HIIMS21] Xuanxiang HUANG, Yacine IZZA, Alexey IGNATIEV, et Joao MARQUES-SILVA. « On Efficiently Explaining Graph-Based Classifiers ». abs/2106.01350, 2021. 6.2.2, 6.3

[HMS23] Xuanxiang HUANG et Joao MARQUES-SILVA. « The Inadequacy of Shapley Values for Explainability ». *ArXiv*, 2023. 4.2.2

[HTF01] Trevor HASTIE, Robert TIBSHIRANI, et Jerome FRIEDMAN. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., 2001. 3.1.1

[IHI+22] Yacine IZZA, Xuanxiang HUANG, Alexey IGNATIEV, Nina NARODYTSKA, Martin C. COOPER, et Joao MARQUES-SILVA. « On Computing Probabilistic Abductive Explanations ». *Int. J. Approx. Reason.*, 159 :108939, 2022. 8.2.3, 8.6, 8.7

[IIMS20] Yacine IZZA, Alexey IGNATIEV, et Joao MARQUES-SILVA. « On Explaining Decision Trees ». *ArXiv*, abs/2010.11034, 2020. 3.2.1, 3.3.3, 3.3.3, 3.3.3, 12, 6, 54, 6.1.2, 6.2, 6.2.1, 8.5.3, 8.7

[IISMS22] Alexey IGNATIEV, Yacine IZZA, Peter J. STUCKEY, et Joao MARQUES-SILVA. « Using MaxSAT for Efficient Explanations of Tree Ensembles ». Dans *AAAI Conference on Artificial Intelligence*, 2022. 8.1

[Il'01] Victor P. IL'EV. « An approximation guarantee of the greedy descent algorithm for minimizing a supermodular set function ». *Discrete Applied Mathematics*, 114(1) :131–146, 2001. 2.3.3, 2.3.3, 2.3.4, 8.3.2, 8.4.1, 10, 8.6

[IMS21] Y. IZZA et J. MARQUES-SILVA. « On Explaining Random Forests with SAT ». Dans *Proc. of IJCAI'21*, pages 2584–2591, 2021. 4.3

[INAMS20a] Alexey IGNATIEV, Nina NARODYTSKA, Nicholas ASHER, et Joao MARQUES-SILVA. « From Contrastive to Abductive Explanations and Back Again ». Dans *International Conference of the Italian Association for Artificial Intelligence*, 2020. 4.3.2, 4.3.2

[INAMS20b] Alexey IGNATIEV, Nina NARODYTSKA, Nicholas ASHER, et Joao MARQUES-SILVA. « On Relating 'Why ?' and 'Why Not ?' Explanations ». *ArXiv*, abs/2012.11067, 2020. 4.1.2, 4.3, 6.3, 6

[INM19] A. IGNATIEV, N. NARODYTSKA, et J. MARQUES-SILVA. « Abduction-Based Explanations for Machine Learning Models ». Dans *Proc. of AAAI'19*, pages 1511–1519, 2019. 4.2.4, 5.1.1

[INMS19] Alexey IGNATIEV, Nina NARODYTSKA, et Joao MARQUES-SILVA. « On Relating Explanations and Adversarial Examples ». Dans *Neural Information Processing Systems*, 2019. 4.3.2

[JMSSS14] Said JABBOUR, Joao MARQUES-SILVA, Lakhdar SAIS, et Yakoub SALHI. « Enumerating Prime Implicants of Propositional Formulae in Conjunctive Normal Form ». Dans *Logics in Artificial Intelligence*, 2014. 6.2.2, 4

[JR20] Kai JIA et Martin RINARD. « Efficient Exact Verification of Binarized Neural Networks ». Dans H. LAROCHELLE, M. RANZATO, R. HADSELL, M.F. BALCAN, et H. LIN, éditeurs, *Advances in Neural Information Processing Systems*, volume 33, pages 1782–1795. Curran Associates, Inc., 2020. 5.1.1

[K14] Erik TRUMBELJ et Igor KONONENKO. « Explaining prediction models and individual predictions with feature contributions ». *Knowledge and Information Systems*, 41 :647–665, 2014. 4.2.2

[KG07] Andreas KRAUSE et Carlos GUESTRIN. « Near-optimal Observation Selection using Submodular Functions. ». pages 1650–1654, 01 2007. 2.2

[KKT03] David KEMPE, Jon KLEINBERG, et Éva TARDOS. « Maximizing the Spread of Influence through a Social Network ». *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 137-146, 07 2003. 2.2

[KKT09] Pushmeet KOHLI, M KUMAR, et Philip TORR. « P³ Beyond : Move Making Algorithms for Solving Higher Order Functions ». *IEEE transactions on pattern analysis and machine intelligence*, 31 :1645–56, 10 2009. 2.2

[KLMT13] F. KORICHE, J.-M. LAGNIEZ, P. MARQUIS, et S. THOMAS. « Knowledge Compilation for Model Counting : Affine Decision Trees ». Dans *Proc. of IJCAI'13*, pages 947–953, 2013. 1, 11, 1

[KMF+17] Guolin KE, Qi MENG, Thomas FINLEY, Taifeng WANG, Wei CHEN, Weidong MA, Qiwei YE, et Tie-Yan LIU. « LightGBM : A Highly Efficient Gradient Boosting Decision Tree ». Dans *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 3149–3157, Red Hook, NY, USA, 2017. Curran Associates Inc. 3.1.3, 3.3.2

[KNTB09] Yoshinobu KAWAHARA, Kiyohito NAGANO, Koji TSUDA, et Jeff A BILMES. « Submodularity Cuts and Applications ». Dans Y. BENGIO, D. SCHUURMANS, J. LAFFERTY, C. WILLIAMS, et A. CULOTTA, éditeurs, *Advances in Neural Information Processing Systems*, volume 22. Curran Associates, Inc., 2009. 2.3.2

[Koh95] Ron KOHAVI. « A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection ». Dans *International Joint Conference on Artificial Intelligence*, 1995. 3.1.2

[LB95] Yann LECUN et Y. BENGIO. « Convolutional Networks for Images, Speech, and Time-Series ». 01 1995. 3.1.3

[LB11] Hui LIN et Jeff BILMES. « A Class of Submodular Functions for Document Summarization ». Dans *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics : Human Language Technologies*, pages 510–520, Portland, Oregon, USA, juin 2011. Association for Computational Linguistics. 2.2

[LDH+90] Yann LECUN, J DENKER, D HENDERSON, R HOWARD, W HUBBARD, et L JACKE. « Imagenet classification with deep convolutional neural networks ». *Advances in neural information processing systems*, 1990. 3.1.3

[LEC+20] S. M. LUNDBERG, G. G. ERION, H. CHEN, A. J. DEGRAVE, J. M. PRUTKIN, B. NAIR, R. KATZ, J. HIMMELFARB, N. BANSAL, et S. LEE. « From local explanations to global understanding with explainable AI for trees ». *Nat. Mach. Intell.*, 2(1) :56–67, 2020. 4.1.1, 7.3.1

[LEL18] Scott M. LUNDBERG, Gabriel G. ERION, et Su-In LEE. « Consistent Individualized Feature Attribution for Tree Ensembles ». *ArXiv*, abs/1802.03888, 2018. 4.2.2

[Lip18] Z. C. LIPTON. « The mythos of model interpretability ». *ACM*, 61(10) :36–43, 2018. 5.1.1, 6, 6.5, 7.1.1

[LKG+07] Jure LESKOVEC, Andreas KRAUSE, Carlos GUESTRIN, Christos FALOUTSOS, Jeanne M. VANBRIESEN, et Natalie S. GLANCE. « Cost-effective outbreak detection in networks ». Dans *Knowledge Discovery and Data Mining*, 2007. 2.2.1

[LL17] S. LUNDBERG et S-I. LEE. « A Unified Approach to Interpreting M(IJCAIodel Predictions ». Dans *Proc. of NIPS'17*, pages 4765–4774, 2017. 4.1.1, 4.2, 4.2.2, 4.2.2, 7.2.1, 7.3.1

[LLM03] J. LANG, P. LIBERATORE, et P. MARQUIS. « Propositional Independence : Formula-Variable Independence and Forgetting ». *Journal of Artificial Intelligence Research*, 18 :391–443, 2003. 5

[LM08] J. LANG et P. MARQUIS. « On propositional definability ». *Artificial Intelligence*, 172(8-9) :991–1017, 2008. 6.2.2

[LM17] J.-M. LAGNIEZ et P. MARQUIS. « An Improved Decision-DNNF Compiler ». Dans *Proc. of IJCAI'17*, pages 667–673, 2017. 6.2.2, 4, 6.4, 8.7

[LRMM15] Benjamin LETHAM, Cynthia RUDIN, Tyler H. MCCORMICK, et David MADIGAN. « Interpretable classifiers using rules and Bayesian analysis : Building a better stroke prediction model ». *ArXiv*, abs/1511.01644, 2015. 4.1.1

[LS17] Edo LIBERTY et Maxim SVIRIDENKO. « Greedy Minimization of Weakly Supermodular Set Functions ». Dans Klaus JANSEN, José D. P. ROLIM, David WILLIAMSON, et Santosh S. VEMPALA, éditeurs, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2017, August 16-18, 2017, Berkeley, CA, USA*, volume 81 de *LIPIcs*, pages 19 :1–19 :11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. 2.3.3, 3, 2.3.4, 8.3.2, 8.4.2, 11, 8.6, 8.7

[LS20] Giovanni Di LEO et Francesco SARDANELLI. « Statistical significance : p value, 0.05 threshold, and applications to radiomics—reasons for a conservative approach ». *European Radiology Experimental*, 4, 2020. 3.2.3

[Mar00] P. MARQUIS. « *Consequence finding algorithms* », volume 5 de *Handbook on Defeasible Reasoning and Uncertainty Management Systems*, Chapitre 2, pages 41–145. Kluwer Academic Publisher, 2000. 6

[MGC+21] João MARQUES-SILVA, Thomas GERSPACHER, Martin C. COOPER, Alexey IGNATIEV, et Nina NARODYTSKA. « Explanations for Monotonic Classifiers ». Dans *Proc. of ICML'21*, pages 7469–7479, 2021. 5.4

[Mil56] G. A. MILLER. « The Magical Number Seven, Plus or Minus Two : Some Limits On Our Capacity For Processing Information ». *The Psychological Review*, 63(2) :81–97, 1956. 3.3.3, 4.1.1, 8.1

[Mil19] T. MILLER. « Explanation in artificial intelligence : Insights from the social sciences ». *Artificial Intelligence*, 267 :1–38, 2019. 3.1.3, 4.1.1, 4.3.2, 6.1.3

[MML14] R. MARTINS, V. M. MANQUINHO, et I. LYNCE. « Open-WBO : A Modular MaxSAT Solver ». Dans *Proc. of SAT'14*, pages 438–445, 2014. 7.3.1

[Mol20] C. MOLNAR. *Interpretable Machine Learning*. Leanpub, 2020. 3.1.3, 3.2.3, 4.1.1, 4.1.4

[Mos17] Hagar MOSAAD. « *ARRIVAL : A zero-player graph game* ». PhD thesis, 08 2017. (document), 4

[MS13] Shashi MITTAL et Andreas SCHULZ. « An FPTAS for optimizing a class of low-rank functions over a polytope ». *Mathematical Programming*, 141, 01 2013. 2.3.3, 9

[MS23] Joao MARQUES-SILVA. « Logic-Based Explainability in Machine Learning ». *ArXiv*, abs/2211.00541, 2023. 4.1.1, 4.1.2, 4.3

[MSGC+20] Joao MARQUES-SILVA, Thomas GERSPACHER, Martin COOPER, Alexey IGNATIEV, et Nina NARODYTSKA. « Explaining Naive Bayes and Other Linear Classifiers with Polynomial Time and Delay ». Dans H. LAROCHELLE, M. RANZATO, R. HADSELL, M.F. BALCAN, et H. LIN, éditeurs, *Advances in Neural Information Processing Systems*, volume 33, pages 20590–20600. Curran Associates, Inc., 2020. 5.4

[Nar18] Nina NARODYTSKA. « Formal Analysis of Deep Binarized Neural Networks ». Dans *International Joint Conference on Artificial Intelligence*, 2018. 5.3.2

[NIPMS18] Nina NARODYTSKA, Alexey IGNATIEV, Filipe PEREIRA, et Joao MARQUES-SILVA. « Learning Optimal Decision Trees with SAT ». Dans *International Joint Conference on Artificial Intelligence*, 2018. 3.2.1

[NKR+18] N. NARODYTSKA, S. Prasad KASIVISWANATHAN, L. RYZHYK, M. SAGIV, et T. WALSH. « Verifying Properties of Binarized Deep Neural Networks ». Dans *Proc. of AAAI'18*, pages 6615–6624, 2018. 5.3.2

[NTP+22] Meike NAUTA, Jan TRIENES, Shreyasi PATHAK, Elisa NGUYEN, Michelle PETERS, Yasmin SCHMITT, Jörg SCHLÖTTERER, Maurice van KEULEN, et Christin SEIFERT. « From Anecdotal Evidence to Quantitative Evaluation Methods : A Systematic Review on Evaluating Explainable AI ». *ACM Computing Surveys*, 2022. 4.1.1

[NW78] George L. NEMHAUSER et Laurence A. WOLSEY. « Best Algorithms for Approximating the Maximum of a Submodular Set Function ». *Math. Oper. Res.*, 3 :177–188, 1978. 2.3.4

[NWF78] George NEMHAUSER, Laurence WOLSEY, et M. FISHER. « An Analysis of Approximations for Maximizing Submodular Set Functions—I ». *Mathematical Programming*, 14 :265–294, 12 1978. 2.3.2, 2.3.2, 2.3.2, 2.3.4, 5, 8.4.2

[Oxl06] James G. OXLEY. « Matroid Theory (Oxford Graduate Texts in Mathematics) ». 2006. 2.1.3

[Pap94] Ch. H. PAPADIMITRIOU. *Computational complexity*. Addison–Wesley, 1994. 1.2, 1.2.3

[PHB+18] Alun David PREECE, Daniel HARBORNE, Dave BRAINES, Richard J. TOMSETT, et Supriyo CHAKRABORTY. « Stakeholders in Explainable AI ». *ArXiv*, abs/1810.00184, 2018. 4.1.3

[PKSR02] Vili PODGORELEC, Peter KOKOL, Bruno STIGLIC, et Ivan ROZMAN. « Decision Trees : An Overview and Their Use in Medicine ». *Journal of medical systems*, 26 :445–63, 11 2002. 3.2

[PS82] Christos H. PAPADIMITRIOU et Kenneth STEIGLITZ. *Combinatorial Optimization : Algorithms and Complexity*. Prentice-Hall, 1982. 1.2

[PVG+11] F. PEDREGOSA, G. VAROQUAUX, A. GRAMFORT, V. MICHEL, B. THIRION, O. GRISEL, M. BLONDEL, P. PRETTENHOFER, R. WEISS, V. DUBOURG, J. VANDERPLAS, A. PASSOS, D. COURNAPEAU, M. BRUCHER, M. PERROT, et E. DUCHESNAY. « Scikit-learn : Machine Learning in Python ». *Journal of Machine Learning Research*, 12 :2825–2830, 2011. (document), 6, 3.2.1, 3.3.3, 3.3.3, 6.4, 7.3.1

[Qui86] J. R. QUINLAN. « Induction of Decision Trees ». *Machine Learning*, 1(1) :81–106, 1986. 3.2.1, 3.2.1, 6

[Rei87] Ray REITER. « A theory of diagnosis from first principles ». *Artificial Intelligence*, 32 :57–95, 1987. 6.3

[Riv87] Ronald L. RIVEST. « Learning Decision Lists ». *Mach. Learn.*, 2 :229–246, 1987. 3.2.2, 40, 2

[RSB18] M. ROBNIK-SIKONJA et Marko BOHANEC. « Perturbation-Based Explanations of Prediction Models ». Dans *Human and Machine Learning*, 2018. 4.1.4

[RSG16] M. T. RIBEIRO, S. SINGH, et C. GUESTRIN. « "Why Should I Trust You ?" : Explaining the Predictions of Any Classifier ». Dans *Proc. of SIGKDD'16*, pages 1135–1144, 2016. (document), 4.1.1, 4.2, 4.2.1, 15, 16, 4.2.1, 7.2.1

[RSG18] M. T. RIBEIRO, S. SINGH, et C. GUESTRIN. « Anchors : High-Precision Model-Agnostic Explanations ». Dans *Proc. of AAAI'18*, pages 1527–1535, 2018. 4.1.1, 4.2, 4.2.3, 4.2.3

[Rud18] Cynthia RUDIN. « Please Stop Explaining Black Box Models for High Stakes Decisions ». *ArXiv*, abs/1811.10154, 2018. 4.1.1

[RW17] Roy RADA et Hayden WIMMER. « Decision Trees and Financial Variables ». *International Journal of Decision Support System Technology*, 9 :1–15, 01 2017. 3.2

[Sat05] Fujishige SATORU.. « Submodular functions and optimization ». *Annals of Discrete Mathematics*, 58, 2005. 30, 5, 33, 2, 2.3.2, 2.3.2

[SCD16] Y. SHEN, A. CHOI, et A. DARWICHE. « Tractable Operations for Arithmetic Circuits of Probabilistic Models ». Dans *Advances in Neural Information Processing Systems 29 : Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 3936–3944, 2016. 6

[SCD18a] A. SHIH, A. CHOI, et A. DARWICHE. « Formal Verification of Bayesian Network Classifiers ». Dans *Proc. of PGM'18*, pages 427–438, 2018. 5.1.3, 6.1.3

[SCD18b] A. SHIH, A. CHOI, et A. DARWICHE. « A Symbolic Approach to Explaining Bayesian Network Classifiers ». Dans *Proc. of IJCAI'18*, pages 5103–5111, 2018. 5.2.2, 5.2.2

[Sch00] Alexander SCHRIJVER. « A Combinatorial Algorithm Minimizing Submodular Functions in Strongly Polynomial Time ». *J. Comb. Theory, Ser. B*, 80(2) :346–355, 2000. 2.2.1

[Sch03] Alexander SCHRIJVER. *Combinatorial Optimization : Polyhedra and Efficiency*, volume B. 01 2003. 2.3.2

[SDV+16] Ramprasaath R. SELVARAJU, Abhishek DAS, Ramakrishna VEDANTAM, Michael COGSWELL, Devi PARIKH, et Dhruv BATRA. « Grad-CAM : Why did you say that ? Visual Explanations from Deep Networks via Gradient-based Localization ». *CoRR*, abs/1610.02391, 2016. 4.1.1

[SF12] R.E. SCHAPIRE et Y. FREUND. *Boosting : Foundations and Algorithms*. MIT Press, 2012. 42

[SG08] Matthew STREETER et Daniel GOLOVIN. « An Online Algorithm for Maximizing Submodular Functions ». Dans D. KOLLER, D. SCHUURMANS, Y. BENGIO, et L. BOTTOU, éditeurs, *Advances in Neural Information Processing Systems*, volume 21. Curran Associates, Inc., 2008. 2.2

[SG13] Laasya SAMHITA et Hans GROSS. « The "Clever Hans Phenomenon" revisited ». *Communicative integrative biology*, 6 :e27122, 11 2013. 1

[Sha51] Lloyd S SHAPLEY. *Notes on the n-Person Game – II : The Value of an n-Person Game*. 1951. 4.2.2

[SK10] Peter STOBBE et Andreas KRAUSE. « Efficient Minimization of Decomposable Submodular Functions ». Dans J. LAFFERTY, C. WILLIAMS, J. SHAWE-TAYLOR, R. ZEMEL, et A. CULOTTA, éditeurs, *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010. 2.2.1

[Sla97] P. SLAVÍK. « A Tight Analysis of the Greedy Algorithm for Set Cover ». *Journal of Algorithms*, 25(2) :237 – 254, 1997. 6.2.1

[SSBD14] Shai SHALEV-SHWARTZ et Shai BEN-DAVID. « Understanding Machine Learning : From Theory to Algorithms », 2014. 3.1.2

[SSSSS10] Shai SHALEV-SHWARTZ, Ohad SHAMIR, Nathan SREBRO, et Karthik SRIDHARAN. « Learnability, Stability and Uniform Convergence ». *J. Mach. Learn. Res.*, 11 :2635–2670, 2010. 5.1.1

[Sto76] Larry J. STOCKMEYER. « The polynomial-time hierarchy ». *Theoretical Computer Science*, 3(1) :1–22, 1976. 14

[Tse68] G.S. TSEITIN. « *On the complexity of derivation in propositional calculus* », Chapitre Structures in Constructive Mathematics and Mathematical Logic, pages 115–125. Steklov Mathematical Institute, 1968. 1.3.1, 6.2.2

[Tur50] Alan M. TURING. « Computing Machinery and Intelligence ». *Mind*, LIX :433–460, 1950. 1.1

[Val79a] L. G. VALIANT. « The Complexity of Computing the Permanent ». *Theoretical Computer Science*, 8 :189–201, 1979. 1.2.4, 15, 16

[Val79b] Leslie G. VALIANT. « The Complexity of Enumeration and Reliability Problems ». *SIAM J. Comput.*, 8 :410–421, 1979. 1.2.4, 1.3.4

[Val84] Leslie G. VALIANT. « A theory of the learnable ». Dans *Symposium on the Theory of Computing*, 1984. 5.1.1

[Vaz13] V.V. VAZIRANI. *Approximation Algorithms*. Springer Berlin Heidelberg, 2013. 2.3

[Viz64] V. VIZING. « On an estimate of the chromatic class of a p-graph (In Russian) ». *(Russian) Diskret Analiz*, 3, 01 1964. 2.3

[Von07] Jan VONDRAK. « *Submodularity in Combinatorial Optimization* ». PhD thesis, Charles University Faculty of Mathematics and Physics, 2007. 2.3.2

[Von08] Jan VONDRAK. « Optimal Approximation for the Submodular Welfare Problem in the Value Oracle Model ». Dans *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, page 67–74, New York, NY, USA, 2008. Association for Computing Machinery. 2.3.4

[Von10] Jan VONDRÁK. « Submodularity and Curvature : The Optimal Algorithm (Combinatorial Optimization and Discrete Algorithms) ». 2010. 2.3.4, 2.3.4

[Whi35] Hassler WHITNEY. « On the Abstract Properties of Linear Dependence ». *American Journal of Mathematics*, 57 :63–87, 1935. 2.1.3

[WMHK19] S. WÄLDCHEN, J. MACDONALD, S. HAUCH, et G. KUTYNIOK. « The Computational Complexity of Understanding Network Decisions ». *CoRR*, abs/1905.09163, 2019. 8.2.1

[WMHK21] Stephan WÄLDCHEN, Jan MACDONALD, Sascha HAUCH, et Gitta KUTYNIOK. « The Computational Complexity of Understanding Binary Classifier Decisions ». *J. Artif. Intell. Res.*, 70 :351–387, 2021. 8.1, 8.7

[WS11] D.P. WILLIAMSON et D.B. SHMOYS. *The Design of Approximation Algorithms*. Cambridge University Press, 2011. 2.3

[ZF17] Zhi-Hua ZHOU et Ji FENG. « Deep Forest : Towards an Alternative to Deep Neural Networks ». Dans *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17, page 3553–3559. AAAI Press, 2017. 3.1.3

[Zho12] Zhi-Hua ZHOU. *Ensemble Methods : Foundations and Algorithms*, volume 14. 06 2012. 3.3

Résumé

Le besoin et la motivation pour l'intelligence artificielle explicable (XAI) peuvent être résumés en deux objectifs principaux : justification et validation. Les explications permettent de justifier les résultats d'un modèle d'apprentissage automatique en fournissant un motif du raisonnement suivi par le modèle. Une fois que les explications sont extraites, il est essentiel de vérifier la validité de ces explications pour s'assurer qu'elles correspondent aux intentions et aux attentes de l'utilisateur du système d'IA considéré.

Dans cette thèse, nous avons développé des méthodes de génération d'explications locales post-hoc. Nous avons commencé par étudier l'intelligibilité computationnelle des classifieurs booléens, caractérisée par leur capacité à répondre aux requêtes d'explicabilité en temps polynomial. Nous avons montré que l'intelligibilité computationnelle des arbres de décision est supérieure à celle de nombreux modèles d'apprentissage automatique, ce qui nous a conduit par la suite à nous focaliser sur les arbres de décision comme modèle d'apprentissage. Nous avons étudié la capacité des arbres de décision à extraire, minimiser et compter les explications abductives et contrastives. Nous avons montré que l'ensemble complet des raisons suffisantes pour une instance peut être de taille exponentielle, rendant ainsi difficile, voire impossible, la génération de l'ensemble complet. De plus, deux raisons suffisantes pour une même instance peuvent différer de manière significative. Pour tenter de synthétiser l'ensemble des raisons suffisantes pour une instance, nous avons introduit les concepts d'attribut nécessaire et d'attribut pertinent, ainsi que le concept de pouvoir explicatif d'un attribut. Pour réduire le nombre d'explications à fournir à l'utilisateur, nous avons aussi développé des modèles de préférence et des algorithmes pour générer des explications exploitant ces préférences. Cette démarche présente plusieurs avantages. En prenant en compte ces préférences, nous pouvons rechercher des explications abductives qui conviennent à l'utilisateur tout en réduisant considérablement leur nombre. Cependant, même quand on se restreint aux explications préférées, les explications abductives peuvent être de trop grande taille pour être interprétables par les humains en raison des limitations cognitives de ces derniers. Nous avons donc abordé le défi de réduire la taille des explications tout en maintenant une probabilité élevée de prédiction exacte. Ce problème est difficile en général, même pour les arbres de décision. Pour surmonter cette difficulté, nous avons exploré l'approximation des explications probabilistes en utilisant le concept de super-modularité. Nous avons développé deux algorithmes gloutons (GA et GD) pour la minimisation super-modulaire. L'efficacité de ces algorithmes dépend de la courbure de la fonction d'erreur non normalisée qui mesure la précision de l'explication. Empiriquement, nous avons montré l'efficacité de ces algorithmes.

Mots-clés: IA explicable, Apprentissage automatique, Optimisation super-modulaire, Arbre de décision, Contraintes, Explication formelle.

Abstract

The need and motivation for Explainable Artificial Intelligence (XAI) can be summarized by two main objectives : justification and validation. Explanations serve to justify the outcomes of a machine learning model by providing a rationale for the model's decision-making process. Once explanations are extracted, it is crucial to validate their validity to ensure they align with the intentions and expectations of the user of the AI system in question that is considered.

In this thesis, we have primarily focused on generation methods for post-hoc local explanations. We started by examining the computational intelligibility of Boolean classifiers, characterized by their ability to address XAI queries in polynomial time. We have shown that the computational intelligibility of decision trees overcomes that of numerous machine learning models, which subsequently led us to focus on decision trees as an machine learning model. We have investigated the capacity of decision trees to derive, minimize, and calculate abductive and contrastive explanations. We have shown that the set of all sufficient reasons for an instance can be exponentially large, making the generation of the entire set difficult, if not impossible. Furthermore, two sufficient reasons for the same instance can differ significantly. In an attempt to synthesize the set of all sufficient reasons for an instance, we have introduced the concepts of necessary features and relevant features, as well as the concept of explanatory power of a feature. To reduce the number of explanations provided to the user, we have also developed preference models and algorithms to generate explanations that leverage user's preferences. This approach offers several advantages. By taking preferences into account, we can search for abductive explanations that are suited to the user, while significantly reducing their quantity. However, even when considering only the preferred ones abductive explanations can be too large to be interpretable by humans due to their cognitive limitations. To address this issue, we have tackled the challenge of reducing the size of explanations while maintaining a high probability of prediction. This problem is generally hard, even for decision trees. To overcome this difficulty, we have explored the approximation of probabilistic explanations using the concept of supermodularity. We have developed two greedy algorithms (GA and GD) for supermodular minimization. The effectiveness of these algorithms depends on the curvature of the non-normalized error function, which measures the accuracy of the explanation. Empirically, we demonstrated the efficiency of these two algorithms.

Keywords: eXplainable AI, Machine Learning, Supermodular optimization, Decision trees, Constraints, Formal explanations.