Ministère de l'enseignement supérieur et de la recherche

Institut National des Sciences Appliquées Toulouse

Département de Génie Mathématiques & Modélisation

Projet de fin d'études

Apprentissage dans le domaine combinatoire et complexité d'échantillonage de différents langages de représentation de préférence

---

Institut ANITI

NGUYEN Ngoc Bao
MMS - GMM
nbnguyen@etud.insa-toulouse.fr

25 novembre 2021

1 Remerciements

Je tiens à remercier toutes les personnes qui ont contribué au succès de mon stage et qui m'ont aidé lors de la rédaction de ce rapport.

Tout d'abord, j'aimerais exprimer mes remerciements sincères à mes tuteurs de stage : Hélène FARGIER et Jérôme MENGIN. Merci Hélène pour ta confiance placée en moi, de m'avoir accepté de réaliser ce stage, et pour les temps précieux que tu m'a donné dans chaque discussion. Merci Jérôme pour ton accompagnement, ton écoute, ton encouragements, le temps de corriger mes démonstrations, et ton aide durant la rédaction du rapport, sans les quelles le résultat final n'est plus possible. J'aimerais vous dire que chaque discussion avec vous a éveillé vraiment mes motivations de travail.

Je souhaitais remercier l'ANITI, l'IRIT et particulièrement l'équipe l'ADRIA qui m'ont donné l'opportunité de travailler dans une environnement très scientifique. Je tiens aussi mes remerciements aux collègues dans le bureau 302 pour vos hospitalités.

Je remercie également le département GMM, le bureau de stage de l'INSA de Toulouse, et particulièrement les membres du jury de mon PFE. Vos engagements dans ce processus sont indispensables pour la validation de mon PFE.

Finalement merci mes amis, H et T, qui m'ont aider de trouver ce stage.

2 Abstrait

L'étude et l'exploitation de préférences sont présences par tout dans notre vie courante, dans plusieurs domaines, notamment économie, psychologie. Cet étude est intéressé récemment par le développement de e-commerce avec la personnalisation des produits. Dans ce projet, nous allons d'abord étudier l'apprentissage de plusieurs langages de représentation de préférence dans le domaine combinatoire : CP-nets, LP-trees, réseaux bayésiens, et GAI-nets sous la définition de l'apprentissage probablement approximativement correct (PAC learning). L'étude se fait notamment sur la complexité d'échantillonage de ces langages, puis nous allons étudier particulièrement la classe des LP-trees linéaires par la généralisation des résultats existants puis proposer quelques algorithmes d'apprentissage de cette classe de LP-trees. Dans la future, nous pourrions faire le recherche sur autre cas non-étudié de autres langages, généraliser le résultat de LP-tree pour le cas non-linéaire et le cas de préférences conditionnelles, et proposer autres algorithmes d'apprentissage.

Mots-clés : préférences conditionnelles, domaine combinatoire, LP-tree, PAC learning, complexité d'échantillonage, algorithme d'apprentissage

Table des matières

Chapitre 1

Introduction

Mon stage au sein de l'ANITI (Artificial and Natural Intelligence Toulouse Institute) et l'IRIT (L'Institut de Recherche en Informatique de Toulouse) dure du 1/6/2021 au 20/11/2021. Il fait partie de la chaire "Knowledge compilation" du programme "IA certifiable" de l'ANITI. L'objectif de la chaire est d'étudier des méthodes de résoudre les problèmes de décision dans le domaine combinatoire concernant les préférences et l'incertitude, ainsi les transformer en approximations plus simples. Ce stage est effectué à l'IRIT, spécifiquement au coeur de l'équipe ADRIA qui fait des études notamment couvrent les domaines du raisonnement, de la décision, de l'argumentation et de l'apprentissage.

Le stage concernent les systèmes de l'aide de décision, par instance les interfaces de configuration en-ligne, qui guide l'utilisateur rapidement aux meilleurs choix possibles à l'aide des informations disponibles accumulées par le système. Certaines modèles de préférences ont été proposés dans des études de l'intelligence artificielle, cela ouvre la possibilité de représenter des préférences bien complexes sur des domaines multi-variables sous une forme assez compacte. Il y a eu des recherches sur les propriétés de certains langages de représentation de préférence. Le but du stage est de compléter ce tableau de propriété. La direction classique est d'étudier la difficulté de requêtes d'information de ces langages (en général, plus un langage a une bonne expressivité, plus sa complexité de requêtes est élevée). Dans ce rapport, afin d'étudier la complexité de requête, on fera la recherche sur la complexité d'échantillonage. On pourrait également étudier autres propriétés importantes de ces langages : l'élicitation de préférence, ou comment ces langages sont appris, etc.

Mon stage est encadré par Dr. Hélène FARGIER, (Directrice de Recherche à CNRS, membre de l'équipe ADRIA-IRIT et porteur principal de la chaire "Knowledge compilation" à l'ANITI), et Dr. Jérôme MENGIN (Responsable du parcours Intelligence Artificielle, Intelligence Collective, Interaction de la spécialité Recherche du M2 à L'université Toulouse III Paul-Sabatier, membre de l'équipe ADRIA-IRIT).

Ce rapport est organisé de manière suivante : La partie 1 et 2 vont établir la notation de préférence dans le domaine combinatoire et introduire certains langages utiles de représentation des préférences. La partie 3 définit l'apprentissage de préférences, introduit la notion de complexité d'échantillonage et décrit le tableau général des résultats existants. Dans la partie 4, nous allons généraliser le cas de LP-tree linéaire afin d'obtenir un résultat plus général. La partie 5 se consiste de 4 nouveaux algorithmes d'apprentissage de k-LP-tree linéaire et la comparaison de ces algorithmes.

Finalement, le code d'expérimentation du partie 5 est disponible à :

`https://github.com/ngocbao1022/linear-k-LP-tree-learning.git`

Chapitre 2

Cadre et objectifs du stage

1 La notation de préférences

Dans la vie courant, nous rencontrons très souvent les situations où on choisit ou on préfère un objet à un autre, par instance, le KFC au Burber King, le Whisky au Vodka, un CPU AMD à un CPU Intel,... cela est la préférence.

Pourtant, très régulièrement la préférence est plus complexe que un choix binaire. Considérons les choix dans un système configurable à plusieurs composants, par exemple dans un tour PC, on a toujours les choix du CPU, du module RAM, de la carte mère, de la carte graphique, de l'alimentation,... Dans cette configuration, le choix de chaque composant dépend d'un ou plusieurs autres composants : un carte mère B450 ne va jamais avec un CPU Intel, une carte graphique puissante est de préférence groupée avec un CPU aussi puissant, le choix de l'alimentation dépend fortement de l'ensemble de tous les autres composants. On appellera cette dépendance la préférence conditionnelle.

Définition 1.1 (Relation Binaire)**.** *Une relation binaire $\mathcal{R}$ sur un ensemble fini A est définit comme un sous-ensemble du produit cartésien $A \times A$, qui est considéré comme l'ensemble des paires ordonnées $(a, b) \in A \times A$. On écrit $a \mathcal{R} b$ si le couple (a, b) appartient à $\mathcal{R}$ et $a \neg\mathcal{R} b$ sinon.*

Le fait que la relation binaire est une relation entre deux objets convient à établir la définition de préférence : la préférence qu'on rencontre consiste souvent à comparer une ou plusieurs paires d'objets.

Définition 1.2 (Propriétés d'un relation binaire)**.** *Une relation binaire $\mathcal{R}$ est dite :*
Réflexible *si $a \mathcal{R} a \ \forall a \in A$*
Complète *si $a \mathcal{R} b$ ou $b \mathcal{R} a \ \forall a, b \in A$*
Antisymétrique *si $a \mathcal{R} b$ et $b \mathcal{R} a \Rightarrow a = b \ \forall a, b \in A$*
Symétrique *si $a \mathcal{R} b \Rightarrow b \mathcal{R} a \ \forall a, b \in A$*
Asymétrique *si $a \mathcal{R} b \Rightarrow b \neg\mathcal{R} a \ \forall a, b \in A$*
Transitive *si $a \mathcal{R} b$ et $b \mathcal{R} c \Rightarrow a \mathcal{R} c \ \forall a, b, c \in A$*

Préordre

Définition 1.3 (Préordre)**.** *Un préordre est une relation binaire qui est **réflexive** et **transitive**.*

Préordre complet

Définition 1.4 (Préordre complet)**.** *Un préordre complet est une relation binaire qui est **réflexive**, **transitive** et **complète**.*

Ordre total

Définition 1.5 (Ordre total)**.** *Un ordre total est une relation binaire qui est* ***réflexive, transitive, complète*** *et* ***antisymétrique****.*

Exemple : La relation "$\geq$" est un ordre total sur l'ensemble des nombres entiers $\mathbb{N}$

Ordre partiel

Définition 1.6 (Ordre partiel)**.** *Un ordre partiel est une relation binaire qui est* ***réflexive, transitive*** *et* ***antisymétrique****.*

2 Différents langages de préférences

Ce tableaux suivant représente des notations souvent utilisées dans ce projet, notamment les notations du domaine combinatoire.

Notations	Significations
n	nombre de variables/attributs
X	un attribut
$\underline{X}$	le domaine du attribut X
x	une valeur de l'attribut X, autrement dit $x \in \underline{X}$
$\mathcal{X}$	l'ensemble des attributs
$\underline{\mathcal{X}}$	le domaine de l'ensemble de tous les attributs, égaux au produit cartésien des domaines des tous les attributs
o	un objet, une instantiation de $\mathcal{X}$, autrement dit $o \in \underline{\mathcal{X}}$
$\succ_X$	une relation d'ordre sur le domaine d'attribut X
$\succ_{\mathcal{X}}$	une relation d'ordre sur le domaine de l'ensemble de tous les attributs
$\succ_{\mathcal{L}}$	une relation d'ordre représentée par le modèle $\mathcal{L}$
S	l'ensemble des données
$\mathcal{H}$	l'ensemble d'hypothèses
$\mathcal{R}$	une relation binaire ou un ordre
$rank(\mathcal{R}, o)$	le rang de l'objet o selon l'ordre total $\mathcal{R}$

FIGURE 2.1 – Les notations utilisés

2.1 CP net

Conditional preference networks ou CP-nets est un langage de représentation adapté aux préférences *ceteris paribus*. Un CP-net contient deux parties : un graphe et des table de préférences conditionnelles associées.

Définition 2.1 (Table de préférence conditionnelle)**.** *Soit X, P deux ensembles d'attributs disjoints de X. Une table de préférence conditionnelle associe à tout $p \in \underline{\boldsymbol{P}}$ un ordre total $\succ_X$ sur le domaine $\underline{\boldsymbol{X}}$, noté $CPT_X(p)$*

Définition 2.2 (Règle de préférence)**.** *Soit $CPT_X(p)$ une table de de préférences conditionnelles sur $\underline{P}$. Chaque paire $(p, CPT_X(p))$ est appelée une règle, et sera noté :*

$$p : CPT_X(p)$$

Si $P = \emptyset$, la table de préférences conditionnelles se réduit à une seule règle.

Définition 2.3 (CP-net)**.** *Un CP-net sur les attributs $\mathcal{X} = \{X_1, ..., X_n\}$ est une graphe dirigé avec n nœuds, chaque nœud étant étiqueté par un attribut différent. Chaque nœud est associé à une table de préférence conditionnelle CPT_X, qui associe à chaque instanciation $\boldsymbol{u}$ des parents de X_i un ordre total sur les valeurs de X_i*

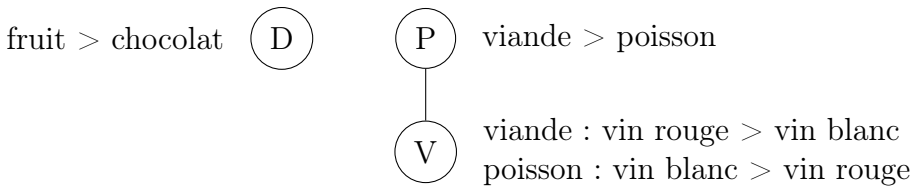


FIGURE 2.2 – Exemple de CP-net

Sémantique des CP-nets

Dans un CP-net, une paire d'objets (o, o') qui ne diffère que par un valeur d'un attribut est appelé un **swap**. Autrement dit, il existe $X \in \mathcal{X}$, $u \in \underline{\mathcal{X} \setminus \{X\}}$ et $x, x' \in \underline{X}, x \neq x'$ tel que $o = \mathbf{u}x$ et $o' = \mathbf{u}x'$.

Un CP-net $\mathcal{C}$ ordonne un swap de manière suivant : dans la table de préférence conditionnelle du nœud X, il y a une seule règle compatible avec $\mathbf{u}$ ordonnant toutes les valeurs de X. On dit que $\mathbf{u}x \succ_{\mathcal{C}} \mathbf{u}x'$ quand $x \succ_X x'$.

Les règles dans une table de préférence conditionnelle dépend de valeur des parents, un arc d'un nœud X_1 au nœud X_2 signifie que la préférence sur X_2 dépend de la valeur de X_1.

TCP-nets

TCP-nets est une variation de CP-nets dans lesquels nous y ajoutons la relation d'importance entre les attributs

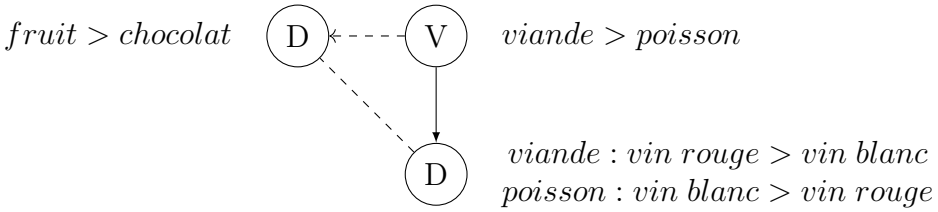


FIGURE 2.3 – Exemple de TCP-net

2.2 LP tree

Définition 2.4 (LP-tree)**.** *LP-tree est une arbre composée de deux parties :*

- *un arbre enraciné indiquant l'importance des attributs. Chaque nœud de cet arbre est étiqueté par un attribut. Aucun attribut ne peut apparaître deux fois dans une branche. Un nœud étiqueté peut avoir soit zéro arête sortante (une feuille), soit une arête sortante sans étiquette, soit plusieurs arêtes sortantes étiquetée par différentes valeurs de ce noeud.*
- *les tables de préférence conditionnelles contenant un ensemble de règle CPT_N pour chaque nœud N.*

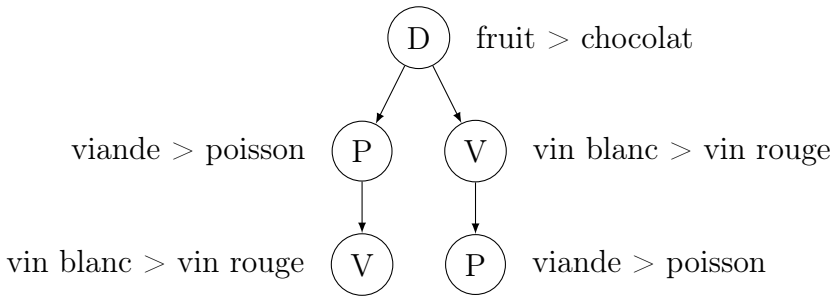


FIGURE 2.4 – Exemple de LP-tree

Définition 2.5 (LP-tree complet)**.** *Soit $\mathcal{L}$ un LP-tree. Une branche de $\mathcal{L}$ est dite complète si et seulement si elle contient tous les attributs de $\mathcal{X}$. Si toutes les branches d'un LP-tree sont complètes, le LP-tree est dit complet.*

LP-tree linéaire

Définition 2.6 (LP-tree linéaire)**.** *Un LP-tree linéaire est un LP-tree avec une unique branche et dont chaque table de préférences conditionnelles ne contient qu'un seule règle.*

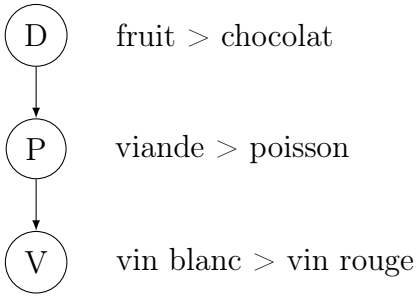


FIGURE 2.5 – Exemple de LP-tree linéaire

k-LP-tree

Définition 2.7 (k-LP-tree)**.** *Un k-LP-tree est une généralisation des LP-tree autorisant l'étiquetage d'un nœud par un ensemble X d'attribut de talle maximale k. Cet ensemble d'attributs est considéré comme un seul attribut à plusieurs dimension, les règles dans les table de préférences conditionnelles définissent un ordre sur le produit cartésien $\underline{X}$ des domaines des attributs dans X.*

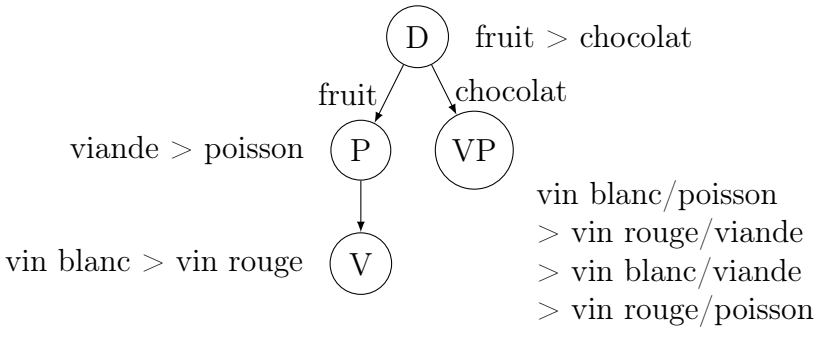


FIGURE 2.6 – Exemple de 2-LP-tree

Sémantique d'un LP-tree

La relation entre deux objets dans un LP-tree se retrouve de manière "top-down", c'est-à dire qu'on compare les objets depuis la racine jusqu'aux feuilles si nécessaire. Prenons la figure 2.6 comme exemple :

- On compare le menu *o : plat viande/ vin blanc/ dessert fruit* avec le menu *o' : plat poisson/ vin blanc/ dessert chocolat*. Dès la racine de l'arbre, *dessert fruit* est préféré à *dessert fruit*, on peut conclure immédiatement que $o \succ_{\mathcal{L}} o'$ sans comparer autres attributs. En effet, tous les objet dans la branche *fruit* sont préféré aux objets dans la branche *chocolat*.
- On compare le menu *o : plat viande/ vin blanc/ dessert fruit* avec le menu *o' : plat poisson/ vin blanc/ dessert fruit*. Cette fois-ci les deux objets appartient à la même branche *dessert fruit*, la règle au nœud P dit que *plat viande* $\succ_P$ *plat poisson*, cela nous permet de conclure que $o \succ_{\mathcal{L}} o'$ sans comparer les choix du vin.
- On compare le menu *o : plat viande/ vin rouge/ dessert chocolat* avec le menu *o' : plat poisson/ vin blanc/ dessert chocolat*. Dans ce cas, deux objets appartient à la branche *dessert chocolat*, puis on compare ces deux objet à la feuille *VP* qui est un nœud à deux attributs. La règle au *VP* dit que *plat poisson/ vin blanc* $\succ_{VP}$ *plat viande/ vin rouge*, donc $o' \succ_{\mathcal{L}} o$.

2.3 Réseaux bayésiens

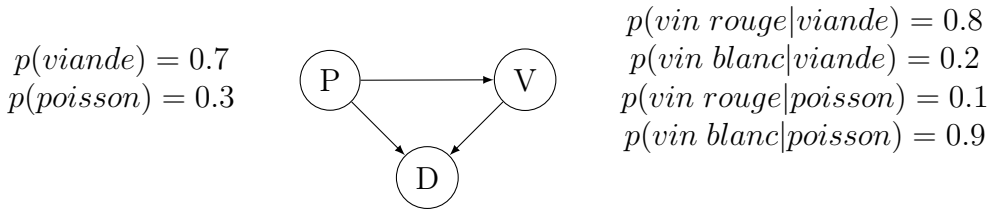
Définition 2.8 (Réseaux bayésien)**.** *Un réseau bayésien sur l'ensemble d'attributs $\mathcal{X}$ est une paire $(\mathcal{G}, \Theta)$ où*

- *$\mathcal{G}$ est une graphe orienté acyclique dont les sommets les attributs de $\mathcal{X}$*
- *Θ est un ensemble de loi probabilité conditionnelles : pour chaque attribut X, $\Theta(X|\boldsymbol{Parent}(X))$ dénote la loi de probabilité de X sachant les valeurs des parents de X.*

Les réseaux bayésiens sont couramment utilisé dans le domaine d'apprentissage automatique car ils peuvent représenter des lois probabilistes bien complexes qui dépendent de plusieurs variables. Chaque observation de l'ensemble des variable peut se associer à une probabilité. Cela ouvre une opportunité de l'appliquer dans l'apprentissage de préférence de manière suivante : on définit que $o \succ o'$ si et seulement si $p(o) > p(o')$.

Sémantique de réseau bayésien

Nous faisons un exemple de choisir la préférence dans l'exemple de figure 2.7.



$$p(fruit|vin\ rouge, viande) = 0.7$$
$$p(chocolat|vin\ rouge, viande) = 0.3$$
$$p(fruit|vin\ blanc, viande) = 0.2$$
$$p(chocolat|vin\ blanc, viande) = 0.8$$
$$p(fruit|vin\ rouge, poisson) = 0.4$$
$$p(chocolat|vin\ rouge, poisson) = 0.6$$
$$p(fruit|vin\ blanc, poisson) = 0.7$$
$$p(chocolat|vin\ blanc, poisson) = 0.3$$

FIGURE 2.7 – Exemple de réseau bayésien

Afin de comparer deux menus *o : viande/vin rouge/chocolat* et *o' : poisson/vin blanc/fruit*, on calcule la probabilité de chaque menu :

$$\begin{aligned} p(o) &= \Theta(viande) \times \Theta(vin\ rouge|viande) \times \Theta(chocolat|vin\ rouge, viande) \\ &= 0.7 \times 0.8 \times 0.3 = 0.168 \end{aligned}$$

$$\begin{aligned} p(o') &= \Theta(poisson) \times \Theta(vin\ blanc|poisson) \times \Theta(fruit|vin\ blanc, poisson) \\ &= 0.3 \times 0.9 \times 0.7 = 0.189 \end{aligned}$$

Puisque $p(o) < p(o')$, on dit que le menu o est strictement préféré au menu o'.

2.4 GAI-nets :

Définition 2.9 (Indépendance additive généralisée)**.** *Soient $Z_1, ..., Z_k$ des sous-ensembles de $\mathcal{X}$ tels que $\mathcal{X} = \cup_{i=1}^{k} Z_i$. On dit qu'il y a indépendence additive généralisé entre $Z_1, ..., Z_k$ pour un ordre total $\mathcal{R}$ si et seulement s'il existe des fonctions réelles $u_i : Z_i \mapsto \mathbb{Z}$ telle que la fonction u définie par :*

$$u(o) = \sum_{i=1}^{k} u_i(o[Z_i]) \forall o \in \underline{X}$$

soit strictement croissante par rapport à $\mathcal{R}$. C'est alors de composition GAI pour u.

Définition 2.10 (GAI-net)**.** *Soient $Z_1, Z_2, ..., Z_k$ des sous-ensemble de $\mathcal{X}$ tels que $\mathcal{X} = \cup_{i=1}^{k} Z_i$ et tels qu'il y ait une indépendance additive généralisée entre $Z_1, Z_2, ..., Z_k$ pour $\mathcal{R}$. Alors un GAI-net représente u est une graphe non orienté $((V, E)$ tel que :*

- *$V = \{Z_1, Z_2, ..., Z_k\}$*
- *Pour chaque arête $\{Z_i, Z_j\} \in E, Z_i \cap Z_j \neq \emptyset$. De plus, pour tout paire de chemin dans G entre Z_i et Z_j dont les nœuds contiennent tous les attributs de T_{ij}*

De plus, chaque arête est étiquetée par l'ensemble non vide $Z_i \cap Z_j$.

Sémantique de GAI-net

Par exemple, on a deux sous-ensembles suivants :

— $Z_1 = \{D, P\}$
— $Z_2 = \{V, P\}$

Les fonction d'utilité u_1 et u_2 sont définie respectivement sur $\underline{Z_1}$ et $\underline{Z_2}$.

$\underline{Z_1}$	chocolat/viande	chocolat/poisson	fruit/viande	fruit/poisson
u_1	10	15	21	26

$\underline{Z_2}$	vin rouge/viande	vin rouge/poisson	vin blanc/viande	vin blanc/poisson
u_2	100	20	27	130

La fonction d'utilité totale $u(o)$ est la composition des fonction d'utilité u_1 et u_2 de manière suivante : $u(o) = u_1(o[Z_1]) + u_2(o[Z_2])$

$$\begin{cases} u(fruit/viande/vinrouge) = 21 + 100 = 121 \\ u(fruit/viande/vinblanc) = 21 + 27 = 48 \\ u(fruit/poisson/vinrouge) = 26 + 20 = 46 \\ u(fruit/poisson/vinblanc) = 26 + 130 = 156 u(chocolat/viande/vinrouge) = 10 + 100 = 110 \\ u(chocolat/viande/vinblanc) = 10 + 27 = 37 \\ u(chocolat/poisson/vinrouge) = 15 + 20 = 35 \\ u(chocolat/poisson/vinblanc) = 15 + 130 = 145 \end{cases}$$

L'ordre totale $\mathcal{R}$ définit par u est comme suivant :

$$fruit/poisson/vin\ blanc \succ chocolat/poisson/vin\ blanc \succ$$
$$fruit/viande/vin\ rouge \succ chocolat/viande/vin\ rouge \succ$$
$$fruit/viande/vin\ blanc \succ fruit/poisson/vin\ rouge \succ$$
$$chocolat/viande/vin\ blanc \succ chocolat/poisson/vin\ rouge$$

Le GAI-net associé est représenté par figure 2.8.

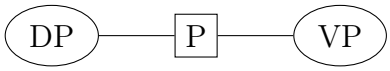


FIGURE 2.8 – Exemple de GAI net

3 Apprentissage PAC et les notations

3.1 Qu-est ce que c'est l'apprentissage automatique ?

L'apprentissage automatique, aussi appelé sous l'anglicisme *machine learning*, est "le processus de convertir expérience en expertise ou connaissance par une machine. Dans ce processus, l'entrée est des données d'apprentissage et la sortie est des connaissances sous forme d'une programme informatique qui peut être exécutée"[8].

L'apprentissage automatique est une notion indispensable de décrire des problèmes dans le domaine d'apprentissage de préférences. La plupart de problèmes dans ce domaine sont des problème *supervisé* où les données sont étiquetées. Dans l'apprentissage de préférence, il y a trois familles d'apprentissage supervisé : l'apprentissage *à partir de comparaisons*, l'apprentissage *à partir d'exemples positifs*, et l'apprentissage *actif*.

La classification est le problème dans lequel les données sont des couples (x, y) où x est une observation et y est la classe d'objet de cet observation. L'objectif est de construire une méthode,à partir d'un ensemble (x_i, y_i) , permettant la prédiction de la classe d'une nouvel observation.

L'apprentissage à partir d'exemples positifs a pour l'objectif de construire un modèle qui représenter une loi de probabilité, à partir d'un jeu d'observations x_i tiré de cette loi.

L'apprentissage actif se base sur le fait que qu'il fait l'interrogation de l'environnement, par instance, il peut demande à l'utilisateur s'il préfère un pull rouge à un pull noir. Dans le problème d'apprentissage actif, aucunes données initiales est disponibles, le but est de apprendre le modèle $\mathcal{M}$ caché en minimisent le nombre des questions.

3.2 Complexité d'échantillonage et l'apprentissage PAC - Probablement Approximativement Correcte

Afin de bien définir l'apprentissage PAC [8] et quel est son sens dans le problème d'apprentissage de préférences, nous introduisons les notions essentielles.

Classification binaire

Il n'y a que deux classes en classification binaire, noté *1* et *0*. Par exemple, la classification de qualité des produits dans les industries peut avoir deux classes *1 "certifié"* et *0 "non-certifié"*.

On fait souvent l'hypothèse qu'il existe une fonction parfaite qui associe à chaque observation sa classe correspondante. L'objectif dans la classification est de apprendre une fonction plus "proche" possible de cette fonction parfaite.

La fonction apprise appartient à l'ensemble des fonction de $\underline{X}$ dans $\underline{Y}$. Un sous ensemble de cet ensemble dans lequel nous tirons la fonction apprise est appelé **l'ensemble d'hypothèses**. Chaque fonction dans cet ensemble est appelé **une hypothèse**. La fonction parfaite qui décrit parfaitement la relation entre $\underline{X}$ et $\underline{Y}$ est appelé **l'hypothèse parfaite**.

Dans l'apprentissage de préférence, le problème d'apprentissage à partir des comparaisons peut être interprété comme un problème de classification binaire car chaque comparaison associe un couple (x_1, x_2) avec une étiquette *1* ou *0* qui peuvent se traduit comme $x_1 \succ x_2$ et $x_2 \succ x_1$ respectivement.

Définition 3.1 (L'hypothèse de "realizability"). *Cet assomption dit que l'hypothèse parfaite appartient à l'ensemble d'hypothèses.*

Minimisation de risque empirique

L'algorithme d'apprentissage prend en l'entrée un ensemble d'observations S tiré selon une distribution $\mathcal{D}$, étiquetées par un fonction cible f. Puisque la distribution $\mathcal{D}$ et la fonction f sont

inconnues, l'erreur absolue est aussi inconnue. Afin d'évaluer l'erreur pendant l'apprentissage, nous définissons alors une **erreur d'apprentissage** de manière empirique :

$$L_S(h) = \frac{|\{i \in \{1, 2, ..., m\} : h(x_i) \neq y_i\}|}{m}$$

où m est le cardinal de S, h est l'hypothèse considérée.

Nous appelons cet erreur souvent **l'erreur empirique** ou **la perte empirique**. Le but de l'apprentissage est normalement de minimiser cette fonction de perte. Une point faible de cet approche est qu'elle peut nous amène à du surapprentissage, cependant nous n'en parlons pas dans la limite de ce rapport.

Apprentissage PAC - Probablement Approximativement Correct

Définition 3.2 (définition PAC)**.** *Une classe d'hypothèses $\mathcal{H}$ est dite "PAC apprenable" s'il existe une fonction $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ et un algorithme d'apprentissage satisfaisant :*

$\forall \epsilon, \delta \in (0,1)^2$, $\forall$ distribution $\mathcal{D}$ sur $\mathcal{X}$, $\forall$ fonction cible binaire $f : \mathcal{B} \to \{0,1\}$, si l'hypothèse de "realizability" (qui dit qu'il existe une hypothèse parfait $h \in \mathcal{H}$) tient, alors :

L'algorithme d'apprentissage sur $m \geq m_{\mathcal{H}}$ exemples i.i.d sur $\mathcal{D}$, étiquetés par f, retourne une hypothèse h telle que : avec probabilité au moins $1-\delta$, la fonction de perte vérifie $L_{(\mathcal{D},f)}(h) \leq \epsilon$.

La fonction $m_{\mathcal{H}}$ dans cette définition est appelé **la complexité d'échantillonage** ou en l'anglicisme **sample complexity**. Cette notation fait une grande partie de la discussion dans cet article.

La dimension de Vapnik–Chervonenkis Dans la recherche de complexité d'échantillonage, la dimension Vapnik-Chervonenkis ou dimension VC est une notion très utile car on a un lien direct entre la dimension VC et la complexité d'échantillonage dans le problème de classification binaire.

Définition 3.3. *La dimension VC de $\mathcal{H}$ est le cardinal du plus grand ensemble C qui peut être pulvérisé par $\mathcal{H}$.*

On dit un ensemble C est pulvérisé par un ensemble d'hypothèse $\mathcal{H}$ quand il existe une hypothèse $h \in \mathcal{H}$ qui sépare tous les *0* des *1*.

Le lien direct entre dimension VC et complexité d'échantillonage sera illustré par les deux théorèmes suivants.

Theorem 3.4 (Théorème fondamental d'apprentissage statistique - livre "Understanding Machine Learning : From Theory to Algorithms" [8])**.** *Dans le cadre d'apprentissage passif, on suppose que la classe d'hypothèses $\mathcal{H}$ des fonction de domaine $\mathcal{X}$ à $\{0,1\}$ a pour $VCdim(\mathcal{H}) = d < \infty$. Alors, il existe des constant C_1, C_2 telle que :*

$\mathcal{H}$ est PAC apprenable avec complexité d'échantillonage :

$$C_1 \frac{d + \log(1/\delta)}{\epsilon} \leq m_{\mathcal{H}}(\epsilon, \delta) \leq C_2 \frac{d \log(1/\epsilon) + \log(1/\delta)}{\epsilon}$$

Theorem 3.5 (Steve Hanneke 2016 [7])**.** *Soit $\mathcal{H}$ un ensemble d'hypothèses dont la dimension VC est finie, alors :*

$$m_{\mathcal{H}}(\epsilon, \delta) = O(\frac{VC(\mathcal{H}) + ln(1/\delta)}{\epsilon})$$

Le problème d'apprentissage à partir d'un ensemble d'exemples positifs

Dans l'apprentissage de préférence à partir d'exemples positifs, les données sont un ensemble d'objets tiré selon une loi de probabilité cachée. Le but est de apprendre un modèle qui représente l'un ordre total ordonné par la probabilité de tirer des objets selon cette loi.

Nous définissons d'abord une fonction $rank(\mathcal{R}, o)$ où $\mathcal{R}$ un ordre total et o un objet appartenant à cette ordre. Cette fonction est simplement le rang de l'objet o dans cet ordre : rang 1 veut dire l'objet est le plus préféré, etc.

En suite, on peut définir une fonction de perte qui s'adapte à ce problème :

Définition 3.6 (ranking loss)**.** *[6] Soit $\mathcal{R}_{cible}$ l'ordre total cible, $\mathcal{R}$ un ordre total quelconque, $p(\boldsymbol{o})$ le probabilité exact de tirer objet $\boldsymbol{o}$ de $\underline{\mathcal{X}}$. Formellement :*

$$rloss_p(\mathcal{R}_{cible}, \mathcal{R}) = \frac{1}{|\underline{\mathcal{X}}|} \sum_{o \in \underline{\mathcal{X}}} p(o)(rank(\mathcal{R}, o) - rank(\mathcal{R}_{cible}, o)$$

On peut montrer que minimiser ce ranking loss revient à trouver un modèle qui minimise l'espérance du rang.

Sachant que la loi de probabilité dans ce ranking loss est cachée et que réduire l'espérance du rang est suffit pour apprendre du modèle, on définit un rang moyen empirique qui s'adapte à ce problème.

Définition 3.7. *Le rang moyen empirique des éléments de $\mathcal{H}$ selon le modèle $\mathcal{L}$ est :*

$$\overline{rank}(\mathcal{L}, \mathcal{H}) = \frac{1}{|\mathcal{H}|} \sum_{o \in \mathcal{H}} rank(\mathcal{L}, o)$$

3.3 L'étude bibliographique sur la complexité d'échantillonage en apprentissage de préférences

En terme de complexité d'échantillonage pour l'apprentissage à partir d'exemples positifs, nous avons $m_{\mathcal{H}} = \frac{2}{\epsilon^2}(\log \frac{1}{\delta} + \log(2n))$pour les LP-trees linéaires démontré par Pierre-François Gimenez[6], et $m_{\mathcal{H}} = O((\frac{1}{\epsilon})^{4/3} \log \frac{1}{\epsilon} \log \frac{1}{\delta} \log \log \frac{1}{\delta})$ pour les réseaux bayésiens, démontré par Nir Friedman et Zohar Yakhini[5].

En terme de complexité d'échantillonage pour l'apprentissage par les comparaisons, il y a un résultat pour CP-net acyclique publié par Alanazi Eisa, Mouhoub Malek, Zilles Sandra [4] qui dit que : $m_{\mathcal{H}} = n(m-1)m^{n-1}$ où n le nombre d'attributs et m la taille du domaine de chaque attribut.

Avec les théorèmes 3.4 et 3.5, il est suffit de chercher les résultats existants sur la dimension VC pour avoir une première estimation sur la complexité d'échantillonage de différents langages de représentation de préférence.

— LP-tree : $VCdim = 2^n - 1$ [3]
— CP-net : $VCdim = m^n - 1$ [4]
— TCP-net : $VCdim = 2^n - 1$ [6]
— Réseaux bayésiens : $VCdim = 2^n - 1$ [6]
— GAI-net : $VCdim = O(2^k n^{k+1})$ [2]

En rassemblant ces résultats et appliquant théorème 3.4, on a la table suivant :

Résultat sur complexité d'échantillonage				
	Comparaisons			Exemples positives
Langages de préférences	VC dim	$m_{\mathcal{H}}$ cadre passif	$m_{\mathcal{H}}$ cadre actif	$m_{\mathcal{H}}$
LP-tree	$2^n - 1$	$O(\frac{2^n - 1 + ln(1/\delta)}{\epsilon})$		$\frac{2}{\epsilon^2}(\log \frac{1}{\delta} + \log(2n))$
CP-net	$m^n - 1$	$O(\frac{m^n - 1 + ln(1/\delta)}{\epsilon})$	$n(m-1)m^{n-1}$	
TCP-net	$2^n - 1$	$O(\frac{2^n - 1 + ln(1/\delta)}{\epsilon})$		
Réseaux Bayésiens	$2^n - 1$	$O(\frac{2^n - 1 + ln(1/\delta)}{\epsilon})$		$O((\frac{1}{\epsilon})^{4/3} \log \frac{1}{\epsilon} \log \frac{1}{\delta} \log \log \frac{1}{\delta})$
GAI-net	$O(2^k n^{k+1})$	$O(\frac{C2^k n^{k+1} + ln(1/\delta)}{\epsilon})$		

4 Complexité d'échantillonage de k-LP-trees

Tandis qu'il existe une résultat sur la complexité d'échantillonage de LP-tree dans le cadre d'apprentissage à partir d'exemples positifs, ce résultat est, en fait, assez limité car il ne donne que la complexité d'échantillonage pour un cas très particulier de LP-tree : 1-LP-tree linéaire avec des attributs binaires. Dans cette partie, nous allons généraliser ce résultat pour les cas que le langage de LP-tree est plus riche, spécifiquement le cas de 1-LP-tree linéaire avec attributs non binaire et le cas de k-LP-tree linéaire.

Notons que dans la suite, nous utilisons souvent le symbole $\mathcal{H}$ dans l'attention de représenter l'ensemble d'exemples positifs, ce qui est différent de l'ensemble d'hypothèse dans la partie précédente.

4.1 Résultats existants

Avec la définition de l'apprentissage à partir d'exemples positifs qu'on a vu dans la partie précédente, Pierre-François Gimenez [6] a démontré que un 1-LP-tree linéaire qui ont l'ordre des nœuds respectant l'ordre du score de la proposition suivante, le 1-LP-tree appris est celui qui minimise le rang moyen empirique.

Proposition 4.1 (Ordre des nœuds dans un LP-tree linéaire optimal)**.** *Soit $\mathcal{H}$ un ensemble d'exemples positifs de $\underline{\mathcal{X}}$ et soit $\mathcal{L}^*$ le LP-tree linéaire qui minimise le rang moyen empirique de $\mathcal{H}$.*

Notons $S(X, \mathcal{H})$ le score de X pour $\mathcal{H}$ défini de la manière suivantes :

$$S(X, \mathcal{H}) = \frac{1}{|\underline{X}| - 1} \sum_{l=1}^{|\underline{X}|-1} l|\mathcal{H}(x_X^{(l+1)})|$$

Alors, pour tout couple d'attributs (X, Y), si la profondeur de X dans $\mathcal{L}^*$ est inférieur à celle de Y, alors :

$$S(X, \mathcal{H}) \leq S(Y, \mathcal{H})$$

Comme nous avons présenté dans la partie précédente, il existe un résultat en l'ordre de grandeurs de complexité d'échantillonage de 1-LP-tree linéaire. Cette conclusion est pourtant fait sous l'hypothèse que tous les attributs sont binaires.

Proposition 4.2. *Soit $\mathcal{X}$ un ensemble d'attributs binaires, L l'ensemble des 1-LP-trees linéaire sur $\mathcal{X}$, soient $\mathcal{L}^* \in L$ the 1-LP-tree linéaire cible, p une distribution de probabilité croissante par rapport à $\mathcal{L}^*$ selon laquelle sont tirés les exemples de $\mathcal{H}$, ϵ et δ deux réels tels que $0 < \epsilon < 1/2, 0 < \delta < 1/2$. Un LP-tree linéaire $\mathcal{L} \in L$ peut être appris à partir d'un ensemble d'exemples $\mathcal{H}$ qui vérifie la propriété suivante :*

$$|\mathcal{H}| \geq \frac{2}{\epsilon^2}(\log 2n - \log \delta) \Rightarrow \mathbb{P}(rloss(\mathcal{L}, \mathcal{L}^*) \geq \epsilon) \leq \delta$$

En terme de signification, cette proposition indique que la complexité d'échantillonage de 1-LP-tree linéaire avec attributs binaires est $O(\frac{2}{\epsilon^2}(\log 2n - \log \delta))$ au sens de l'apprentissage PAC.

4.2 Généralisation pour le cas des attributs non binaires

Afin de refaire la démonstration quand les attributs ne sont pas binaires, nous avons besoins du lemme suivant.

Lemme 4.3. *Soient X un ensemble fini de A éléments, $\gamma > 0$. Soient p et $\hat{p}$ deux applications de A à $\mathbb{R}$ telles que : $|f(x) - \hat{f}(x)| < \gamma \forall x \in A$*

Notons $A = \{x_1, x_2, ..., x_M\}$ triées selon ordre $f(x_1) \geq f(x_2) \geq ... \geq f(x_n)$, puis y_i une permutation des x_i triées selon ordre $\hat{f}(y_1) \geq \hat{f}(y_2) \geq ... \geq \hat{f}(y_n)$.

Alors, pour tout $i \in [1, M]$: $|f(x_i) - f(y_i)| \leq 2\gamma$

Démonstration. Pour chaque $i \in [1, M]$, on va montrer d'abord que $|f(x_i) - \hat{f}(y_i)| < \gamma$

Cas 1 : $\hat{f}(x_i) > \hat{f}(y_i)$

Ce cas sera séparé en deux sous-cas par relation entre $f(x_i)$ et $\hat{f}(y_i)$.

Cas 1.1 : $f(x_i) \geq \hat{f}(y_i)$

Nous avons : $\hat{f}(x_i) > \hat{f}(y_i) \Rightarrow x_i \in \{y_j\}_{j<i}$. Or, $x_i \notin \{x_j\}_{j<i}$, les ensembles $\{y_j\}_{j<i}$ et $\{x_j\}_{j<i}$ sont de même cardinal, il existe donc un élément $z \in \{x_j\}_{j<i} \setminus \{y_j\}_{j<i}$.

Ce nombre z vérifie que $f(z) \geq f(x_i)$ et $\hat{f}(y_i) \geq \hat{f}(z)$. On obtient alors :

$$f(z) \geq f(x_i) \geq \hat{f}(y_i) \geq \hat{f}(z)$$

Sachant que $|f(z) - \hat{f}(z)| < \gamma$, on en déduit que $|f(x_i) - \hat{f}(y_i)| \leq \gamma$.

Cas 1.2 : $f(x_i) < \hat{f}(y_i)$

Dans ce cas, on a $\hat{f}(x_i) > \hat{f}(y_i) > f(x_i)$. Sachant que $|f(x_i) - \hat{f}(x_i)| < \gamma$, on en déduit que $|f(x_i) - \hat{f}(y_i)| < \gamma$.

Cas 2 : $f(y_i) > f(x_i)$

De manière similaire du cas 1, on le sépare en deux sous-cas selon l'ordre entre $\hat{f}(x_i)$ et $f(y_i)$.

Cas 2.1 : $f(x_i) > \hat{f}(y_i)$

$f(y_i) > f(x_i) > \hat{f}(y_i)$. Sachant que $|f(y_i) - \hat{f}(y_i)| < \gamma$, on en déduit que $|f(x_i) - \hat{f}(y_i)| < \gamma$.

Cas 2.2 : $f(x_i) \leq \hat{f}(y_i)$

Nous avons $f(y_i) > f(x_i) \Rightarrow y_i \in \{x_j\}_{j<i}$. Or, $y_i \notin \{y_j\}_{j<i}$, les ensembles $\{x_j\}_{j<i}$ et $\{y_j\}_{j<i}$ sont de même cardinal, il existe donc un élément $z \in \{y_j\}_{j<i} \setminus \{x_j\}_{j<i}$.

Ce nombre z vérifie que $f(x_i) \geq f(z)$ et $\hat{f}(z) \geq \hat{f}(y_i)$. On obtient alors :

$$\hat{f}(z) \geq \hat{f}(y_i) \geq f(x_i) \geq f(z)$$

Sachant que $|f(z) - \hat{f}(z)| < \gamma$, on en déduit que $|f(x_i) - \hat{f}(y_i)| \leq \gamma$.

Cas 3 : $\hat{f}(x_i) \geq \hat{f}(y_i)$ et $f(y_i) \geq f(x_i)$

Dans ce cas, $\hat{f}(y_i) \geq \hat{f}(x_i) > f(x_i) - \gamma$. De plus, $\hat{f}(y_i) \leq f(y_i) + \gamma < f(x_i) + \gamma$. Donc $|f(x_i) - \hat{f}(y_i)| < \gamma$.

Nous avons montré que $|f(x_i) - \hat{f}(y_i)| \leq \gamma$. Ensuite, par l'inégalité triangulaire :

$$\begin{aligned} |f(x_i) - f(y_i)| &\leq |f(x_i) - \hat{f}(y_i)| + |\hat{f}(y_i) - f(y_i)| \\ &< 2\gamma \end{aligned}$$

□

Remark. *Cette démonstration est inspiré directement de la démonstration du lemme 5.5.1 page 106 du thèse de Pierre-François Gimenez [6], la démarche est plus ou moins identique, le résultat est pourtant plus général puis qu'on n'est pas limité dans le cas que p et $\hat{p}$ sont des probabilité et x n'est pas forcement les valeur d'attributs binaires.*

En suite, la proposition suivante estime la complexité d'échantillonage de 1-LP-tree linéaire avec attributs non binaires.

Proposition 4.4. *Soit $\mathcal{X}$ un ensemble d'attributs de domaine taille d, L l'ensemble des LP-trees linéaire sur $\mathcal{X}$, soient $\mathcal{L}^* \in L$ the LP-tree linéaire cible, p une distribution de probabilité croissante par rapport à $\mathcal{L}^*$ selon laquelle sont tirés les exemples de $\mathcal{H}$, ϵ et δ deux réels tels que $0 < \epsilon < 1/2, 0 < \delta < 1/2$. Un LP-tree linéaire $\mathcal{L} \in L$ peut être appris à partir d'un ensemble d'exemples $\mathcal{H}$ qui vérifie la propriété suivante :*

$$|\mathcal{H}| \geq \frac{d^2}{2\epsilon^2}(\log(2n(d-1)) - \log\delta) \Rightarrow \mathbb{P}(rloss(\mathcal{L}, \mathcal{L}^*) \geq \epsilon) \leq \delta$$

Démonstration. Soit $\mathcal{L}$ un LP-tree linéaire. Notons $X((L), i)$ l'attribut à profondeur i dans $\mathcal{L}$ et $x(\mathcal{L}, i, l)$ sa l-ème valeur préféré dans la table de préférence optimale.

Pour chaque objet o, son rang est :

$$rank(\mathcal{L}, o) = d^n - \sum_{i=1}^{n}\sum_{l=0}^{d-1} l.d^{n-i}\mathbb{1}_{o[X(\mathcal{L},i)]=x(\mathcal{L},i,l)}$$

Le rang moyen espéré d'un LP-tree linéaire est :

$$\begin{aligned}
\mathbb{E}_p[rank(\mathcal{L}, o)] &= \sum_{o\in\mathcal{X}} p(o) rank(\mathcal{L}, o) \\
&= \sum_{o\in\mathcal{X}} p(o)(d^n - \sum_{i=1}^{n}\sum_{l=0}^{d-1} l.d^{n-i}\mathbb{1}_{o[X(\mathcal{L},i)]=x(\mathcal{L},i,l)}) \\
&= 1 - \sum_{o\in\mathcal{X}} p(o) \sum_{i=1}^{n}\sum_{l=0}^{d-1} l.d^{n-i}\mathbb{1}_{o[X(\mathcal{L},i)]=x(\mathcal{L},i,l)} \\
&= 1 - \sum_{i=1}^{n}\sum_{o\in\mathcal{X}} p(o) \sum_{l=0}^{d-1} \sum_{o[X(\mathcal{L},i)]=x(\mathcal{L},i,l)} l.d^{n-i} \\
&= 1 - \sum_{i=1}^{n}\sum_{l=1}^{d-1} p(x(\mathcal{L}, i, l)).l.d^{n-i}
\end{aligned}$$

La fonction de perte s'écrit :

$$rloss(\mathcal{L}, \mathcal{L}^*) = \sum_{i=1}^{n} d^{-i} \sum_{l=1}^{d-1} l(p(x(\mathcal{L}, i, l)) - p(x(\mathcal{L}^*, i, l)))$$

Il existe un nombre γ telle que $\forall i \in \{1, ..., n\}\ \forall l \in \{1, ..., d-1\}$: $|p(x(\mathcal{L}, i, l)) - \hat{p}(x(\mathcal{L}, i, l))| < \gamma$ car les fonctions p et $\hat{p}$ sont bornées.

Posons

$$f(\mathcal{L}, i) = \sum_{l=1}^{d-1}(l \times p(x(\mathcal{L}, i, l))$$

$$\hat{f}(\mathcal{L}, i) = \sum_{l=1}^{d-1}(l \times \hat{p}(x(\mathcal{L}, i, l))$$

Nous avons :

$$|f(\mathcal{L}, i) - \hat{f}(\mathcal{L}, i)| \leq \sum_{l=1}^{d-1} |p(x(\mathcal{L}, i, l)) - \hat{p}(x(\mathcal{L}, i, l))| \leq \sum_{l=1}^{d-1} \gamma = \frac{d(d-1)}{2}\gamma$$

On sait que les $f(\mathcal{L}, i)$ avec $i = 1, ..., n$ est une permutation de $f(\mathcal{L}^*, i)$ avec $i = 1, ..., n$ car dans $\mathcal{L}^*$, seulement l'ordre des nœuds change.

Appliquons la lemme 2.1 sur l'ensemble A contenant tous les affectations possibles d'un nœud, on a $f(\mathcal{L}, i) - f(\mathcal{L}^*, i) < 2\frac{d(d-1)}{2}\gamma = d(d-1)\gamma$, donc :

$$\begin{aligned} rloss(\mathcal{L}, \mathcal{L}^*) &\leq \sum_{i=1}^{n} d^{-i} d(d-1)\gamma \\ &= (\sum_{i=1}^{n} d^{-i}) d(d-1)\gamma \\ &= \frac{d^n - 1}{d^n(d-1)} \times d(d-1)\gamma \\ &= \frac{d^n - 1}{d^{n-1}}\gamma \\ &\leq \frac{d^n}{d^{n-1}}\gamma = d\gamma \end{aligned}$$

Posons $\epsilon = d\gamma \Leftrightarrow \gamma = \frac{\epsilon}{d}$

Par l'inégalité de Boole :

$$\begin{aligned} \mathbb{P}(rloss(\mathcal{L}, \mathcal{L}^*) \geq \epsilon) &\leq \mathbb{P}(\bigcup_{X \in \mathcal{X} \text{ et } x \in \underline{X} \setminus \{x_0\}} |p(x) - \hat{p}(x)| \geq \gamma) \\ &= n(d-1) \max \mathbb{P}(|p(x) - \hat{p}(x)| \geq \gamma) \end{aligned}$$

par l'inégalité de Hoeffding :

$$\begin{aligned} &\leq n(d-1).2.\exp(-2N\gamma^2) \\ &= 2n(d-1)\exp(-\frac{2N\epsilon^2}{d^2}) \end{aligned}$$

On veut que $\mathbb{P}(rloss(\mathcal{L}, \mathcal{L}^*) \geq \epsilon) \leq \delta$, il suffit que :

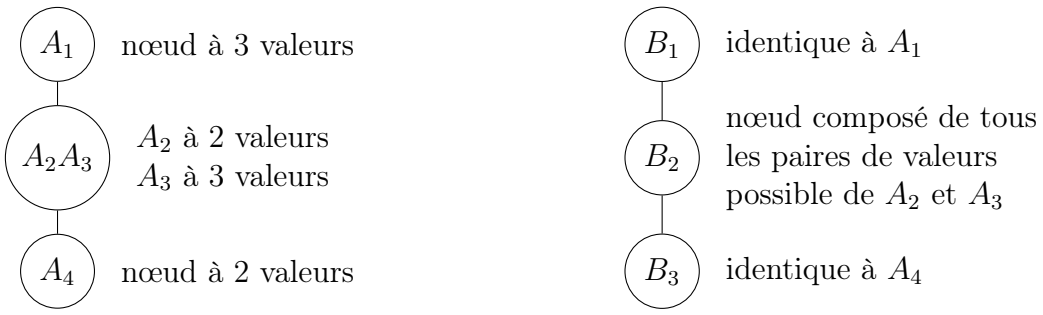
$$\begin{aligned} & 2n(d-1)\exp(-\frac{2N\epsilon^2}{d^2}) \leq \delta \\ \Leftrightarrow & \log(2n(d-1)) - \frac{2N\epsilon^2}{d^2} \leq \log\delta \\ \Leftrightarrow & \frac{2N\epsilon^2}{d^2} \geq \log(2n(d-1)) - \log\delta \\ \Leftrightarrow & N \geq \frac{d^2}{2\epsilon^2}(\log(2n(d-1)) - \log\delta) \end{aligned}$$

On peut conclure a la fin : si $N \geq \frac{d^2}{2\epsilon^2}(\log(2n(d-1)) - \log\delta)$, alors $\mathbb{P}(rloss(\mathcal{L}, \mathcal{L}^*) \geq \epsilon) \leq \delta$. □

Ici, nous disons également que la complexité d'échantillonage est égale à $O(\frac{d^2}{2\epsilon^2}(\log(2n(d-1)) - \log\delta))$.

4.3 Généralisation pour k-LP-tree linéaire

Dans un k-LP-tree linéaire, un nœud à plusieurs attributs est considérer comme un nœud à un seul attribut à plusieurs valeurs. Nous se posons la question si la proposition 4.4 est utile dans cette généralisation pour k-LP-tree.



(a) Exemple de 2-LP-tree linéaire

(b) 1-LP-tree linéaire analogique

FIGURE 2.9 – Analogie entre k-LP-tree linéaire et 1-LP-tree linéaire

Proposition 4.5. *Soit $\mathcal{X}$ un ensemble d'attributs de domaine taille d, L l'ensemble des k-LP-trees linéaire sur $\mathcal{X}$, soient $\mathcal{L}^* \in L$ the k-LP-tree linéaire cible, p une distribution de probabilité croissante par rapport à $\mathcal{L}^*$ selon laquelle sont tirés les exemples de $\mathcal{H}$, ϵ et δ deux réels tels que $0 < \epsilon < 1/2, 0 < \delta < 1/2$. Un k-LP-tree linéaire $\mathcal{L} \in L$ peut être appris à partir d'un ensemble d'exemples $\mathcal{H}$ qui vérifie la propriété suivante :*

$$|\mathcal{H}| = O(\frac{d^{2k}}{2\epsilon^2}\log(\frac{n^k(d^k-1)}{\delta})) \Rightarrow \mathbb{P}(rloss(\mathcal{L}, \mathcal{L}^*) \geq \epsilon) \leq \delta$$

Démonstration. Dans l'approche à ce problème, on va sur-estimer la complexité d'échantillonnage par estimer celle d'un LP-tree linéaire plus complexe.

Dans un k-LP-tree, chaque nœud possède au maximum k attributs. Cela veut dire qu'on a au maximum $d_c = d^k$ modalités dans chaque nœud.

De plus, puisqu'il y a maximum k attributs dans chaque nœud, on peut considérer que chaque nœud ne contient qu'un seul attribut "combiné" tiré dans l'ensemble des attributs "combinés" de maximum k attributs. Cet ensemble des attributs "combiné" contient tous les combinaisons de maximum k parmi n attributs, il contient alors $n_c = \sum_{i=1}^{k} C_n^i$ attributs "combinés".

Considérons un 1-LP-tree linéaire L_c de n nœuds, chaque nœud a d^k modalités. Cette arbre peut contenir toutes les relations ordinaires représentées dans notre k-LP-tree linéaire cible. Un ensemble d'exemples de taille N permet l'apprentissage de L_c à confiance $1-\delta$ va aussi permettre l'apprentissage de notre k-LP-tree à confiance $1-\delta$. Selon la proposition 2.2, un nombre N satisfaisant l'inégalité suivant permet cet apprentissage :

$$N \geq \frac{d_c^2}{2\epsilon^2}(\log(2n_c(d_c-1)) - \log\delta) = \frac{(d^k)^2}{2\epsilon^2}(\log(2(\sum_{i=1}^{k} C_n^i)(d^k-1)) - \log\delta)$$

Sachant que $C_n^i = \frac{\prod_{j=0}^{i-1}(n-j)}{\prod_{j=1}^{i} j} = \Theta(n^i)$, on en déduit que $\sum_{i=1}^{k} C_n^i = \Theta(n^k)$. Donc

$$\frac{(d^k)^2}{2\epsilon^2}(\log(2(\sum_{i=1}^{k} C_n^i)(d^k-1)) - \log\delta) = \Theta(\frac{d^{2k}}{2\epsilon^2}\log(\frac{n^k(d^k-1)}{\delta}))$$

Cela entraîne que $N = O(\frac{d^{2k}}{2\epsilon^2}\log(\frac{n^k(d^k-1)}{\delta}))$ □

Remark. *Ce résultat est une sur-estimation de la complexité d'échantillonage car dans en pratique, le k-LP-tree n'a jamais plus de n nœuds et si le nombre maximum de modalités est atteint (k attributs à ce nœud) à un ou plusieurs nœuds, le nombre total des nœuds est encore moins que n.*

5 Algorithme pour k-LP tree avec attributs binaires

Nous rappelons que dans la définition de apprentissage PAC, nous faisons l'hypothèse qu'il existe une algorithme d'apprentissage assez qui retourne la critère de perte assez petite avec une probabilité définie. Parlant de la fonction perte, avec le méthode de calcule du complexité d'échantillonage pour un 1-LP-tree linéaire, il est plus facile de borner le résultat quand l'algorithme apprendre le k-LP-tree linéaire optimal que dans le cas l'algorithme n'apprendre pas l'arbre optimale. En effet, le calcul dans le premier cas ne comprend que le calcul de borner la fonction perte par les probabilités de tirer le bon nombre d'exemples pour chaque attribut. À l'autre côtés, le seconde cas nous demande de manipuler en même temps ces probabilités et l'écart entre le meilleur k-LP-tree linéaire que l'algorithme peut apprendre et le k-LP-tree linéaire optimal.

Pour pouvoir bien établir un résultat sur la complexité d'échantillonage de k-LP-tree linéaire, nous optons pour la première route, c'est-à-dire de chercher l'algorithme retournant l'arbre optimale.

5.1 Les k-LP trees classiques

5.1.1 L'algorithme naïf apprenant k-LP tree linéaire

Algorithm 1: L'algorithme naïf d'apprentissage de k-LP tree linéaire

Data: L'ensemble d'exemples positifs H
L'emsemble A des attributs A_i, domaine D_i
Result: Un k-LP tree linéaire
$noeuds \leftarrow$ Tous les noeuds de taille maximum k consiste d'attributs appartenant au A
$H_{freq} \leftarrow$ list de nombre d'occurrences de chaque élément unique de H
for $noeud \in noeuds$ **do**
  $Score_i \leftarrow Score(noeud, H_{freq})$
end
$O \leftarrow$ liste ordonnée par $Score_i$ croissante des noeud dans $noeud$
$Attrs \leftarrow$ l'emsemble vide
$tree \leftarrow$ liste vide
$i \leftarrow 1$
while $Attrs \neq A$ **do**
  $new\_attr \leftarrow noeud_i \cap Attrs$
  if $new\_attr = noeud_i$ **then**
    $tree \leftarrow tree + noeud_i$
    $Attrs \leftarrow Attrs \cup noeud_i$
  end
  $tree \leftarrow tree + noeud_i$
  $i \leftarrow i + 1$
end
Descendre $tree$ et calculer les tables de préférence à chaque nœud selon H_{freq}
$PT \leftarrow$ les tables de préférence correspondance
return *tree et ses tables de préférence PT*

Cet algorithme naïf imite l'algorithme d'apprentissage de LP-tree linéaire à nœuds simples en calculant l'ordre des nœuds et prendre le premier arbre qu'on peut construire à partir de cette ordre.

On se demander si cet algorithme peut donner l'arbre optimal en sortie comme c'est le cas avec les 1-LP-trees linéaires. Une procédure de test est à être proposée pour répondre à cette question. Ensuite, si les tests sont positifs, il faut penser au preuve, sinon on cherche la raison

et on essaie à s'améliorer l'algorithme.

5.1.2 Le test pour l'algorithme naïf

Pour le test, nous générons un ensemble d'exemples positifs aléatoire pour 4 attributs binaire puis faisons une recherche exhaustive de l'arbre optimal sur l'ensemble de 3-LP tree sur ces 4 attributs. Après, on va comparer l'arbre résultant avec l'arbre retourné par l'algorithme 1. Pour la recherche exhaustive, afin d'avoir la liste de tous les k-LP tree à partir d'un ensemble d'attributs, nous avons proposé l'algorithme suivante :

Algorithm 2: L'algorithme donnant toutes les séquences représentatives des k-LP-tree à n attributs

```
Data: n le nombre d'attributs
k la taille max de chaque nœud
Result: Tous les séquences représentatives des k-LP tree à n attributs
liste_arbre ← liste contenant un arbre vide
for i ← 1 → n do
    nouvelle_list ← liste vide
    for tree ∈ liste_arbre do
        m ← nombre des nœuds dans tree
        for j ← 0 → m do
            nouvelle_list ← nouvelle_list + tree ajouté de nœud i à position j comme
              un nouveau nœud.
        end
        for j ← 1 → m do
            if nœud j est de taille < k then
                nouvelle_list ← nouvelle_list + tree augmenté de l'attribut i dans le
                  nœud j.
            end
        end
    end
    liste_arbre ← nouvelle_list
end
return liste_arbre
```

Proposition 5.1. *L'algorithme 2 retourne la liste des tous les séquences représentative des k-LP-tree à n attributs.*

Démonstration. L'algorithme se fait par l'ajout de nouveau attribut dans tous les k-LP-tree existants. Donc on va procéder par récurrence.

Hypothèse : $liste\_arbre$ à n attributs est tous les k-LP-trees possibles.

Initialisation : $n = 1$, le seul k-LP-tree est un nœud simple.

Récurrence : supposons que $liste\_arbre(n)$ contient tous les k-LP-trees à n attributs.

On appelle $\mathcal{T}$ l'ensemble de tous les k-LP-trees à $n+1$ attributs. Prenons un élément $\mathcal{L} \in \mathcal{T}$. En supprimant l'attribut $n+1$ dans $\mathcal{L}$, nous obtenons un LP-tree dans lequel chaque nœud est de taille max k aussi car la suppression n'augmente jamais la taille des nœuds. Le LP-tree résultant est un k-LP-tree à n attributs donc appartient au $list\_arbre(n)$. L'opération inverse de cette suppression, autrement dit l'ajout dans la position supprimée, nous donne exactement le k-LP-tree au début. C'est-à-dire tous les éléments de $\mathcal{T}$ peut être construit à partir d'un élément de $list\_arbre(n)$.

Il nous reste à montrer que l'algorithme insère un nœud $n+1$ à toutes les positions possible dans chaque élément de $list\_arbre(n)$. En effet, la création d'un nouveau noeud contenant seulement $n+1$ et l'ajout de $n+1$ dans un nœud existant sans dépasser la limite de taille sont les seules façons d'ajouter l'attribut $n+1$ dans un k-LP-tree existant.

Cela nous permet de conclure que tous les élément de $\mathcal{T}$ peut être construits à partir de $list\_arbre(n)$. Donc $\mathcal{T} \subset list\_arbre(n+1)$. L'hypothèse est vraie pour $n+1$.

Conclusion : l'hypothèse est vraie pour tous $n \in \mathbb{N}$ □

Dans cette proposition, on peut même démontrer que chaque séquence dans $list\_arbre$ est unique, la conclusion est pourtant assez forte pour notre procédure de test.

Une fois que la génération des tous les k-LP-trees est possible, nous pouvons lancer le test pour comparer le minimum dans la recherche exhaustive et le k-LP-tree appris par l'algorithme 1.

Nous avons espéré que cet algorithme peut donner l'arbre optimal en sortie pourtant les expériences réalisées nous montrent que ce n'est pas toujours le cas. En fait on a vu assez souvent que dans plusieurs cas, l'arbre apprise n'est pas l'optimal (il est quand même parmi les meilleurs).

5.1.3 L'explication pour la non-optimalité de l'algorithme naïf

Nous étions étonnants que cet algorithme n'apprendre pas l'optimal car intuitivement le procédure de choisir l'arbre à partir de l'ordre des nœuds est presque le même et que le score utilisé pour les ordonner peut s'adapter pour le cas de nœud à multiples attributs. Afin de bien comprendre la raison, il nous faut re-établir les éléments de base dans ce cas de nœuds à multiple attributs.

La fonction score est tout à fait compatible avec un nœud à n'importe quelle taille car si le nœud a plusieurs attributs, nous pouvons le traiter comme un grand nœud à plusieurs modalités. Faute de problème posé par le score, nous allons examiner la proposition 4.1 dans le cas k-LP-tree.

Proposition 5.2 (Généralisation de proposition 4.1 pour k-LP tree)**.** *Soit $\mathcal{H}$ un ensemble d'exemples positifs de $\underline{\mathcal{X}}$, soit $P_{\mathcal{X}}$ une partition de $\mathcal{X}$ dont les élément ont une taille maximum k, $\mathcal{N}$ l'ensemble des nœuds multiples correspondant à cette partition des attributs, soit $\mathcal{L}^*$ le k-LP tree linéaire qui minimise le rang moyen empirique de $\mathcal{H}$ parmi les k-LP tree constitués de nœuds dans $\mathcal{N}$.*

Notons $S(X, \mathcal{H})$ le score de X pour $\mathcal{H}$ défini de la manière suivantes :

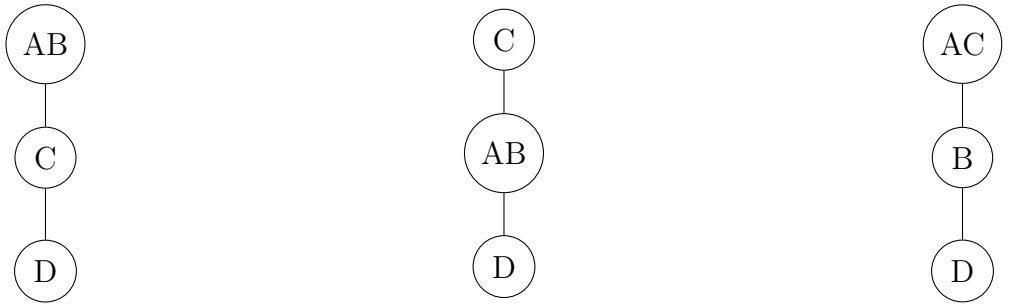
$$S(X, \mathcal{H}) = \frac{1}{|\underline{X}| - 1} \sum_{l=1}^{|\underline{X}|-1} l|\mathcal{H}(x_X^{(l+1)})|$$

Alors, pour tout couple de nœuds (X, Y), si la profondeur de X dans $\mathcal{L}^*$ est inférieur à celle de Y, alors :

$$S(X, \mathcal{H}) \leq S(Y, \mathcal{H})$$

Démonstration. Cette proposition est un corollaire directe de la proposition 4.1 en traitant les nœuds multiple comme des nœuds simple à plusieurs valeurs. □

Ayant généralisé cette proposition, nous voyons que ce score ne nous permet de comparer que les k-LP-tree possèdant les même nœuds. Par exemple, supposons qu'on a un ordre de score : $S(AB) < S(AC) < S(C) < S(B) < S(D) < ...$



(a) Arbre 1 (b) Arbre 2 (c) Arbre 3

FIGURE 2.10 – Exemples

Selon la proposition 5.2, l'arbre 1 est est mieux que l'arbre 2 tandis que ce score nous donne aucune information sur la comparaison entre l'arbre 1 et l'arbre 3. On peut dire que la classe des k-LP-tree possèdent les nœuds (AB, C, D) n'est pas comparable à la classe des k-LP-tree possèdent les nœuds (AC, B, D) en utilisant seulement ce score $S(X, \mathcal{H})$.

5.1.4 Une fixe pour l'algorithme naïf

Cela nous donne un algorithme pour garantir le recherche de k-LP-tree optimal : on sait que la recherche du meilleur k-LP-tree possèdant des nœuds prédéfinis est réalisé avec l'algorithme 1, il nous reste à comparer directement le rang moyen empirique de ces meilleurs k-LP-trees des classes incomparables afin de trouver le meilleur arbre ultime. La composition des deux algorithmes dans la suite nous permettrons d'apprendre le k-LP-tree linéaire optimal.

Algorithm 3: L'algorithme retournant tous les séquences de noeuds sans répétitions d'attributs, respectant une ordre total prédéfinie

Data: La liste ordonnés des noeuds $O : noeud_1 > noeud_2 > ... > noeud_m$ de longeur m
Les attributs contenant dans chaque noeud
Result: Tous les séquences des noeuds sans répétitions, respectant l'ordre
`TOUS-SOUS-SEQUENCES`$(S, O(m))$
if $m = 1$ **then**
  $S \leftarrow noeud_1$
  $list_S \leftarrow list_S + S$
else
  for $i \leftarrow 1$ **to** m **do**
    $S \leftarrow S + noeud_i$
    $O' \leftarrow O$
    for $attr \in noeud_i$ **do**
      $O' \leftarrow$ supprimer tous les noeuds qui contient $attr$ dans O'
    end
    $list_S \leftarrow list_S +$ `TOUS-SOUS-SEQUENCES`(S, O')
  end
end
return $list_S$

Remark. *On note ici que l'algorithme 3 peut retourner la liste de tous les séquences sans répétition des attributs respectant une ordre totale donnée mais elle ne s'assure pas l'apparition des tous les attributs. Il nous faut encore tester si la séquence des nœuds est éligible pour la*

constitution d'un arbre à n attributs. Sachant le besoin de ce test, nous allons l'implémenter directement dans l'algorithme d'apprentissage suivant.

Algorithm 4: L'algorithme d'apprentissage de k-LP-tree linéaire optimal à partir de l'ensemble d'exemples positifs

Data: L'ensemble d'exemples positifs H
L'ensemble A des attributs A_i, domaine D_i
Result: Le k-LP tree linéaire optimal
$noeuds \leftarrow$ Tous les noeuds de taille maximum k consiste d'attributs appartenant au A
$H_{freq} \leftarrow$ list de nombre d'occurrences de chaque élément unique de H
for $noeud \in noeuds$ **do**
  $Score_i \leftarrow Score(noeud, H_{freq})$
end
$O \leftarrow$ liste ordonnée par $Score_i$ croissante des noeud dans $noeud$
$tree\_list \leftarrow$ Algorithme_2($sequence\_vide, O$) (l'algo 3)
$min\_rang\_moyen \leftarrow \infty$
for $tree \in tree\_list$ **do**
  if *tree contient tous les attr$_i$ donnés* **then**
    Descendre $tree$ et calculer les tables de préférence à chaque nœud selon H_{freq}
    $rang\_moyen\_tree \leftarrow$ le rang moyen empirique de $tree$ selon H_{freq}
    if $min\_rang\_moyen > rang\_moyen\_tree$ **then**
      $min\_rang\_moyen \leftarrow rang\_moyen\_tree$
      $j \leftarrow$ indice de cette arbre dans $tree\_list$ $PT \leftarrow$ les tables de préférence correspondance
    end
  end
end
$optimal\_tree \leftarrow tree\_list[j]$
return *optimal_tree et ses tables de préférence PT*

Tandis que cet algorithme d'apprentissage donne l'arbre optimal, elle n'est pas efficace, c'est-à-dire elle ne se fait pas dans temps polynomial. La calcule du rang moyen empirique est la plus coûteuse et on doit l'exécute M fois avec M le nombre des séquences éligibles. On peut montrer que le nombre M correspond au nombre des partitions d'un ensemble de taille n en sous-ensembles de taille maximale k. En effet, pour chaque partition de ce type, il existe toujours un ensemble unique des nœuds éligibles à constituer un k-LP-tree et vice versa. En suite, chaque ensemble des nœuds éligibles correspond à une calcule de rang moyen empirique car il existe un ordre unique de nœuds respectant l'ordre total de tous les nœuds possible.

Afin de montrer que l'algorithme est coûteuse, nous pouvons donner un minorant du nombre M et montrer que ce nombre croît très vite en fonction de n. On propose que :

$$M > \sum_{i=1}^{k} C_n^{n-i\lfloor \frac{n}{i} \rfloor} \frac{(i\lfloor \frac{n}{i} \rfloor)!}{(i!)^{\lfloor n/i \rfloor} \lfloor \frac{n}{i} \rfloor!}$$

$\frac{(i\lfloor \frac{n}{i} \rfloor)!}{(i!)^{\lfloor n/i \rfloor} \lfloor \frac{n}{i} \rfloor!}$ est en fait le nombre de partitions d'un ensemble de $i\lfloor \frac{n}{i} \rfloor$ éléments en sous-ensembles de taille exactement i [1], et $C_n^{n-i\lfloor \frac{n}{i} \rfloor}$ le nombre de choix des $n - i\lfloor \frac{n}{i} \rfloor$ éléments parmi n qui reste dans le cas n n'est pas divisible par i.

Ce nombre n'est qu'un minorant car dans l'apprentissage, on considère aussi des cas où les nœuds n'est pas de taille fixée : quelques nœuds de taille 1, quelque nœuds de taille 2 ou 3, etc. Ces cas ne sont pas compté dans le seconde membre.

n\k	1	2	3	4
1	1	2	3	4
2	1	2	3	4
3	1	4	5	6
4	1	4	8	9
5	1	16	26	31
6	1	16	26	41
7	1	106	176	211
8	1	106	386	421
9	1	946	1226	1541
10	1	946	3746	5321
11	1	10396	25796	31571
12	1	10396	25796	31571
13	1	135136	335336	410411
14	1	135136	1536536	2062061
15	1	2027026	3428426	6056051

FIGURE 2.11 – Seconde membre en fonction de n et k

Afin de mettre en perspective, seulement le terme $\frac{(i\lfloor\frac{n}{i}\rfloor)!}{(i!)^{\lfloor n/i\rfloor}\lfloor\frac{n}{i}\rfloor!}$ quand n est divisible par i est $\frac{n!}{(i!)^{n/i}(\frac{n}{i})!}$. En fonction de n avec i fixé, puisque $(\frac{n}{i})!$ est réglable par rapport à $n!$, il est un factoriel de n divisé par un exponentiel de n donc qu'il croît de manière plus vite que l'exponentiel. Dans notre expérimentations (dont on va parler dans la partie 5.3), tandis que l'apprentissage d'un modèle à 4 attributs se fait quasi-instantanément, il prends presque 3 heures pour apprendre un k-LP-tree à 10 attributs.

Cet algorithme n'est donc pas efficace.

5.2 Les k-LP trees avec répétition aux attributs binaires

Pendant la recherche de l'algorithme apprenant le k-LP tree optimal, nous normalement permettre pas de répétition des attributs dans une arbre. Pourtant nous avons constaté que en permettant la répétition des attributs, nous aurons une nouvelle façons de représenter la préférence conditionnelle.

Définition 5.3 (k-LP tree linéaire avec répétition)**.** *Un k-LP tree linéaire avec répétition est un k-LP tree linéaire où chaque attribut peut apparaître plusieurs fois par sa existence dans différents nœuds. La première apparition d'un attribut dans l'arbre nous donne l'information sur la préférence des valeurs de cet attribut, les apparitions suivantes s'interprète comme la condition dans une préférence conditionnelle.*

Étant donnée le LP-tree avec préférence conditionnelle est un langage plus riche que celui sans préférence conditionnelle, l'apprentissage de ce type d'arbre est plus difficile. Le résultat peut ne pas être optimal, pourtant probablement meilleur que même l'arbre optimal de la classe de LP-tree sans préférence conditionnelle. Dans la suite, on va découvrir quelques algorithmes d'apprentissage de ce types de LP-tree.

5.2.1 L'algorithme naïf

Algorithm 5: L'algorithme naïf apprenant k-LP tree linéaire avec répétition

Data: L'ensemble d'exemples positifs H
L'emsemble A des attributs A_i et ses domaine D_i
Result: Un k-LP tree linéaire avec répétition
$noeuds \leftarrow$ Tous les noeuds de taille maximum k consiste d'attributs appartenant au A
$H_{freq} \leftarrow$ list de nombre d'occurrences de chaque élément unique de H
for $noeud \in noeuds$ **do**
  $Score_i \leftarrow Score(noeud, H_{freq})$
end
$O \leftarrow$ liste ordonnée par $Score_i$ croissante des noeud dans $noeud$
$Attrs \leftarrow$ l'emsemble vide
$tree \leftarrow$ liste vide
$i \leftarrow 1$
while $Attrs \neq A$ **do**
  $new\_attr \leftarrow noeud_i \cap Attrs$
  if $new\_attr \neq \emptyset$ **then**
    $tree \leftarrow tree + noeud_i$
    $Attrs \leftarrow Attrs \cup noeud_i$
  end
  $i \leftarrow i + 1$
end

Remark. *Cet algorithme naïf utilise la même fonction score que celui utilisé dans l'apprentissage de LP-tree linéaire pour ordonner les nœuds. Ce score n'est pourtant pas le score exact de ces nœuds car une fois qu'un nœud peut avoir des préférences conditionnelles, les rangs locaux des valeurs dans ce nœuds sont dépendant de valeurs données aux parents. Ce score peut donc être considéré comme une approximation du score des nœuds. Nous rappelons ici que le but ici n'est pas de trouver un algorithme parfait. Un algorithme qui ne retourne pas toujours l'optimal n'est pas forcement un algorithme mauvais. On va pourtant faire de corrige afin de s'améliorer la précision du algorithme.*

5.2.2 L'algorithme naïf corrigé

L'apprentissage naïf est réalisé par le calcul naïf du score. Pendant les expériences, nous avons constaté que l'algorithme naïf peut apprendre des arbre assez éloignés de la cible à cause de la différence entre la formule du score utilisée dans l'apprentissage et la formule du score utilisée dans l'évaluation de rang. Intuitivement, ce premier score ne prend en compte que la rang global des objets tandis que le deuxième score se calcule à partir du rang local conditionnel des objets dans chaque nœud.

Proposition 5.4 (Formule du rang exact d'un objet dans un LP-tree linéaire total)**.** *Soit $\mathcal{L}$ un k-LP tree linéaire avec répétition, m son nombre des nœuds, $\mathcal{X}$ l'ensemble des attributs,*

$$rank(\mathcal{L}, o) = 1 + \sum_{i=0}^{m}((\sum_{l=0}^{|\underline{\mathcal{X}}(\mathcal{L},i)-1|} l \times (\sum_{k=0}^{|\underline{\mathcal{C}}_i|-1} [C_i = k][o[X(\mathcal{L}, i)] = x^{(l+1)}_{X(\mathcal{L},i,C_{ik})}])) \prod_{j=i}^{m} \frac{|\underline{\mathcal{X}}(\mathcal{L}, j)|}{|\underline{\mathcal{C}}_j|})$$

Définition 5.5 (Score corrigé pour préférence conditionnelle)**.**

$$S^*(X, \mathcal{H}) = \frac{1}{|\underline{X}| - 1} \sum_{l=1}^{|\underline{X}|-1} l \times (\sum_{k=0}^{|\underline{\mathcal{C}}_i|-1} [C_i = k]|\mathcal{H}(x^{(l+1)}_{X(\mathcal{L},i,C_{ik})})|))$$

Remark. *On remarque ici que ce score est identique à celui de la proposition 4.1 si l'ensemble de conditions $\mathcal{C}_i$ a une seule valeur, autrement dit s'il n'y a pas de condition.*

Algorithm 6: L'algorithme naïf corrigé apprenant un k-LP tree linéaire avec répétition

Data: L'ensemble d'exemples positifs H
L'ensemble A des attributs A_i et ses domaine D_i
Result: Un k-LP tree linéaire avec répétition
$noeuds \leftarrow$ Tous les noeuds de taille maximum k consiste d'attributs appartenant au A
$H_{freq} \leftarrow$ list de nombre d'occurrences de chaque élément unique de H
$Attrs \leftarrow$ l'emsemble vide
$tree \leftarrow$ liste vide
$i \leftarrow 1$
while $Attrs \neq A$ **do**
  for $noeud \in noeuds$ **do**
    $Score_i \leftarrow Score^*(noeud, H_{freq})$
  end
  $best\_noeud \leftarrow min(Score_i)$ $new\_attr \leftarrow best\_noeud \cap Attrs$
  if $new\_attr \neq \emptyset$ **then**
    $tree \leftarrow tree + noeud_i$
    $Attrs \leftarrow Attrs \cup noeud_i$
  end
  $noeuds \leftarrow noeuds \setminus \{best\_noeud\}$
  $i \leftarrow i + 1$
end

5.3 Expérimentations

Ces expérimentations mesurent empiriquement l'efficacité des algorithmes proposés à réapprendre un arbre cachée. La méthodologie de expérimentation est identique à celle du chapitre 5.6 dans le thèse de référence [6], le protocole se consiste aussi de 3 étapes :

— Générer aléatoirement $\mathcal{L}$, un k-LP-tree ou un k-LP-tree avec répétition
— Simuler un ensemble d'exemples positifs à partir d'une loi probabilité p décroissant par rapport à $\succ_{\mathcal{L}}$
— Apprendre un k-LP-tree de type correspondante à partir de cet ensemble d'exemples positifs

5.3.1 Protocoles

Nous allons faire deux tests : un pour les arbres à $n = 4$ attributs et un pour les arbres à $n = 10$ attributs. La génération de k-LP-tree classique et la génération de k-LP-tree avec répétition se fait de manières différentes. Tous les deux sont générés de manière *top-down* (de la racine jusqu'aux feuilles) et k est fixé à 3. Pour un k-LP-tree classique, la taille de chaque nœuds est tirée aléatoirement selon un loi uniforme entre 1 et 3, les attributs appartenant à ce nœud sont tirés aléatoirement dans l'ensemble d'attributs absents de l'ensemble d'ancêtres de ce nœud, la table de préférence correspondante est une permutation de tous les valeurs possibles de ce nœud. Pour un k-LP-tree avec répétition, chaque nœud est tiré aléatoirement a partir de l'ensemble de tous les noeuds possibles avec la contraint qu'au moins un nouvel attribut apparaît dans ce nœud, la table de préférence est aussi une permutation de tous les valeurs possibles.

Une fois que l'arbre est généré, on aura un ordre total sur l'ensemble des objets. Les nombres d'exemples des objets seront tirés à partir d'une distribution géométrique tronquée : un objet de rang r sera tiré avec une probabilité $p(r) = K\mu(1-\mu)^{r-1}$ avec μ un constant choisi. Dans nos expérimentations, on fixe $\mu = 0.3$.

Dans l'évaluation du *ranking loss*, on aimerait estimer la *risque réelle* au lieu de la *risque empirique* (rang moyen empirique) car la *risque réelle* représente bien la vraie distance entre l'arbre appris et l'arbre cible. Ce risque réelle peut être représenté par :

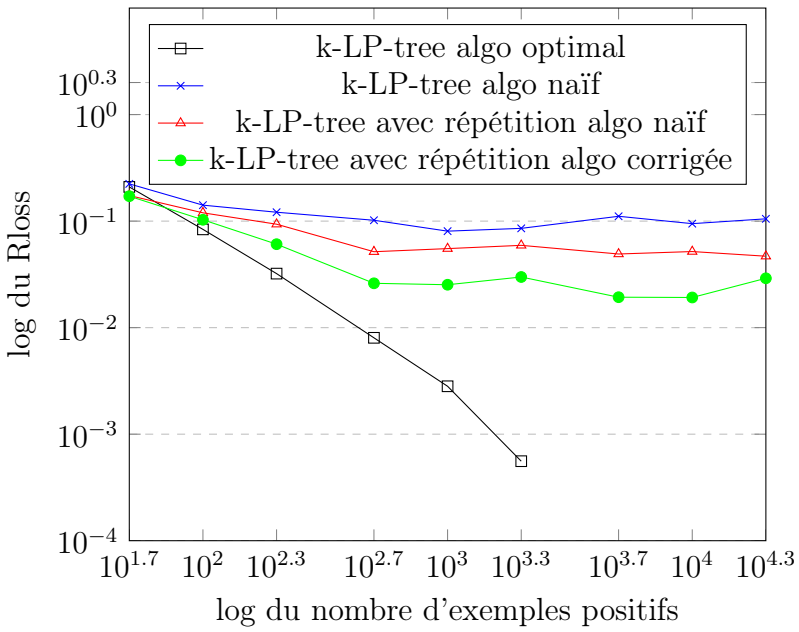
$$rloss_p(\mathcal{R}_{cible}, \mathcal{R}) = \frac{1}{|\underline{\mathcal{X}}|}\mathbb{E}_p[rank(\mathcal{R}, .)] - \frac{1}{|\underline{\mathcal{X}}|}\mathbb{E}_p[rank(\mathcal{R}_{cible}, .)]$$

Il est donc nécessaire de tirer aléatoirement l'ensemble d'exemples positifs selon le rang des objets afin de calculer le ranking loss réel par un méthode Monte-Carlo. Dans notre protocole, par la limite du temps de calcule, le ranking loss réel de chaque arbre est calculé à partir de 2000 ensembles d'exemples positifs et chaque point dans les graphes de la partie suivante représente 200 arbres aléatoirement générées.

5.3.2 Résultats

La première expérimentation compare 4 algorithmes d'apprentissage qu'on a proposées pour un $n = 4$ assez petit et $k = 3$. La figure 2.12 compare ces 4 algorithmes en terme de *ranking loss* entre arbre simulée et arbre apprise.

D'abord, on peut observer une tendance générale de décroissance de *ranking loss* quand le nombres d'exemple augmente. Les algorithmes naïfs et l'algorithme corrigé pour k-LP-tree avec répétition ont pourtant des fluctuation du rloss moyen à la fin, ce qui veut dire que à partir d'un grandeur du nombre d'exemples positifs, ces algorithmes n'apprend plus d'information. Dans ce cas de $n = 4$, $k = 3$, ce nombre est environ 1000. A l'autre côtés, l'algorithme optimal a un moyen *ranking loss* bien mieux que les autres algorithmes et à partir de 5000 exemples positifs, *rloss* entre l'arbre cible et l'arbre appris vaut zéro dans tous les cas aléatoires générés. Cela confirme que avec assez de d'exemples, cet algorithme apprend presque toujours le meilleur.



L'algo optimal a $rloss = 0$ à partir de $x = 5000$

FIGURE 2.12 – Rloss en fonction de nombre d'exemples, $n = 4$

Dans la deuxième expérimentation, on a fixé $n = 10$ et $k = 3$, dans ce cas, nous n'avons pas pu tester l'algorithme optimal car il n'est pas efficace, le coût de calcule croît très vite en fonction de n et k. Nous avons fait une essai afin de estimer le temps de calcule : tandis que à $n = 4$ l'algorithme optimal donne le résultat quasi-instantanément, il a pris environ 10500 secondes = 2 heures 55 minutes pour $n = 10$. Cela confirme notre assertion en complexité temporelle de cet algorithme. Vu que l'algorithme n'est pas efficace et que chaque point dans le graphe représente 200 arbres aléatoirement générées et ré-apprises, nous avons décidés de ne pas tester l'algorithme optimal dans ce cas.

Pour les 3 autres courbes, nous voyons encore une fois la tendance décroissante de *ranking loss* en fonction du nombre d'exemples. L'algorithme corrigée reste mieux que l'algorithme naïf pour k-LP-tree avec répétition, cela confirme que le score corrigé s'améliore la capacité de retenir d'informations contenant dans l'ensemble d'exemples positifs.

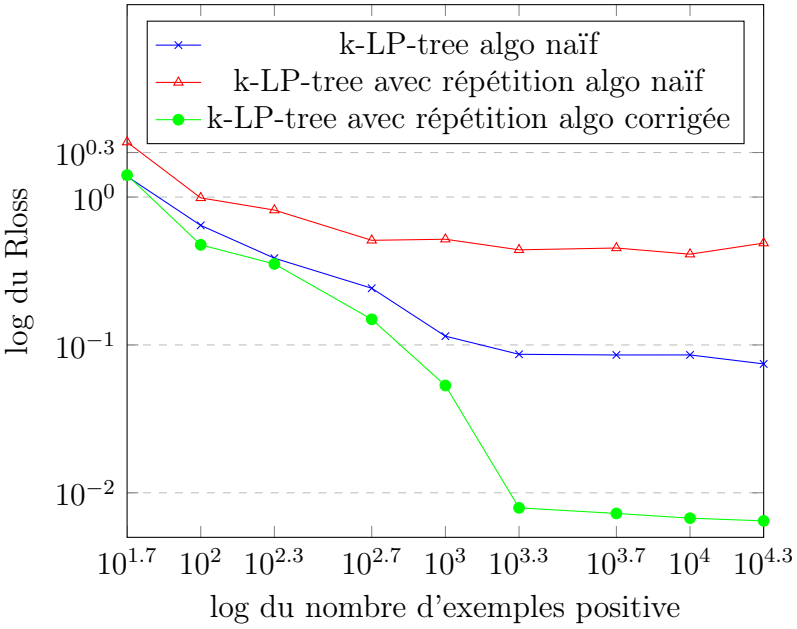


FIGURE 2.13 – Rloss en fonction de nombre d'exemples, $n = 10$

Chapitre 3

Conclusions et perspectives

Conclusions

L'apprentissage de préférence dans le domaine combinatoire est un sujet récemment intéressé par les études à l'aide de son application potentielle dans plusieurs domaine, notamment l'économie. Afin de contribuer à ce champ d'études, nous nous avons concentré sur l'étude de complexité d'échantillonage de différents langages de représentation des préférences au sens d'apprentissage probablement approximativement correct (PAC).

Notre première étape était une recherche documentaire dans laquelle on a vu qu'il existe certains mode d'apprentissage de références : apprentissage à partir de comparaisons, apprentissage à partir d'exemples positifs, et apprentissage actif. Nous avons vu aussi que la complexité d'échantillonage est en général différent dans chaque cas spécifique, c'est-à-dire pour chaque langage de représentation et pour chaque mode d'apprentissage, il faut une taille différente de l'ensemble d'apprentissage afin de pouvoir bien apprendre un modèle. À l'aide des résultats existants sur la dimension VC et le lien entre la dimension VC et la complexité d'échantillonage, nous avons pu donner un ordre grandeur de cette complexité pour tous les langage concerné dans ce rapport dans le cas d'apprentissage à partir d'exemples positifs. Dans le cas des autres modes d'apprentissage, nous avons fait une recherche simple des résultats existants.

Nous souhaitions compléter les cas non-étudiés et généraliser des résultats obtenus dans des cas spécifiques. Pour cela, on s'est concentré sur le LP-tree pour la suite car ses résultats étudiés était fait pour une classe de LP-tree très limité dans l'expressivité du langage : LP-tree linéaire à attributs binaires. Nous avons généralisé pas à pas ce résultat par passer à attributs non-binaires, puis passer au cas k-LP-tree. Des algorithme d'apprentissage de k-LP-tree linéaire avons été également proposés. Pendant cette recherche, nous nous sommes tombé sur une nouvelle classe de k-LP-tree qui peut représente la préférence conditionnelle d'un nouvelle manière. Des algorithmes d'apprentissage pour cette nouvelle classe d'arbre étaient aussi proposés.

Perspectives

Ce stage était une expérience absolument originale dans mon projet professionnel. Le fait que j'ai réalisé le stage dans un institut académique m'a permis d'apprendre et raffiner plusieurs compétences, ainsi d'avoir une vision générale sur le monde académique.

Le première recherche documentaire était assez simple, elle m'a demandé de s'habituer aux nouvelles notions, de comprendre le jargon dans un domaine spécifique et de comprendre le contenu d'un article. Grâce aux études à l'INSA, ces activités n'étaient pas pénible, pourtant quand le nombre d'article recherché était assez grand, ça me pose une nouvelle question : comment organiser tous ces références et bien noter les points importants (ou potentiellement importants). Heureusement il existe des logiciels de productivité qui aident à gérer tous ces informations.

Pendant l'essai de généraliser la proposition en complexité d'échantillonage de LP-tree,

j'étais bloqué au lemme 4.3 en essayant reformulant le quantité du seconde membre de l'inégalité au cas où les attributs ne sont pas binaires. Plusieurs idées ont été proposées, une tentative de généraliser à l'aide de récurrence en taille de domaine d'attributs a été essayée. Jusqu'à ce qu'on ne considère plus que les x_i sont des valeurs des attributs et qu'on fasse des graphes d'exemples qui représente les fonctions de probabilité, j'ai réalisé que le seconde membre de l'inégalité reste toujours 2γ et les étapes de démonstration reste presque identique que celles de la version originale. C'était une expérience unique, j'ai appris que même quand le travail demandé a une solution assez simple, le chemin de qui nous y amène est tellement original.

Dans le deuxième partie de généralisation de résultat sur LP-tree, j'ai eu la chance de essayer l'implémentions des algorithmes d'apprentissage de LP-tree. À mon avis, ce travail était très rafraîchissant sachant que je restais dans la partie théorique pendant 3-4 mois précédents.

Annexe A

Plaquette de présentation de la société

Ce stage est financé par l'ANITI et effectué au sein de l'équipe ADRIA de l'IRIT.

1 ANITI

Ces informations suivantes peut se trouver sur le site `https://aniti.univ-toulouse.fr/` :

ANITI, **Artificial and Natural Intelligence Toulouse Institute**, est l'institut interdisciplinaire d'intelligence artificielle de Toulouse. L'activité d'ANITI repose sur 3 grands piliers : **recherche scientifique**, **formation** et **contribution au développement économique**.

Sa spécificité est de développer une nouvelle génération d'**intelligence artificielle** dite hybride, associant de façon intégrée des techniques d'apprentissage automatique à partir de données et des modèles permettant d'exprimer des contraintes et d'effectuer des raisonnements logiques.

Rassemblant environ 200 chercheur.e.s issu.e.s des universités, écoles d'ingénieurs et organismes de recherche scientifique et technologique de Toulouse et sa région, et d'une cinquantaine de partenaires. Les secteurs d'application stratégiques ciblés sont la mobilité et les transports et la robotique/cobotique pour l'industrie du futur.

L'enjeu majeur est de favoriser les échanges entre les industriels qui adressent leurs problématiques aux scientifiques et aux académiques pour développer de nouveaux marchés, de nouvelles applications technologiques.

ANITI fait partie, avec Grenoble, Nice et Paris, des quatre instituts interdisciplinaires d'intelligence artificielle (**3IA**) qui sont mis en place pour une durée de 4 ans renouvelable dans le cadre du **Programme Investissements d'avenir du plan Villani**. Ces instituts travailleront en réseau avec pour objectif de faire de la France un des leaders mondiaux en intelligence artificielle.

2 IRIT

On peut trouver les informations suivantes sur le site `https://www.irit.fr/` :

L'Institut de Recherche en Informatique de Toulouse (IRIT), une des plus imposantes Unité Mixte de Recherche (UMR 5505) au niveau national, est l'un des piliers de la recherche en Occitanie avec ses 600 membres, permanents et non-permanents, et une centaine de collaborateurs extérieurs. De par son caractère multi-tutelle (CNRS, Universités toulousaines), son impact scientifique et ses interactions avec les autres domaines, le laboratoire constitue une des forces structurantes du paysage de l'informatique et de ses applications dans le monde du numérique, tant au niveau régional que national. Notre unité a su, par ses travaux de pointe et

sa dynamique, définir son identité et acquérir une visibilité incontestable, tout en se positionnant au cœur des évolutions des structures locales : Université de Toulouse, ainsi que les divers dispositifs issus des investissements d'avenir (LabEx CIMI, IRT Saint-Exupéry, SAT TTT...).

Notre recherche se structure autour de **cinq grands sujets scientifiques** :

1. Conception et construction de systèmes (fiables, sûrs, adaptatifs, distribués, communicants, dynamiques...)
2. Modélisation numérique du monde réel
3. Concepts pour la cognition et l'interaction
4. Etude des systèmes autonomes adaptatifs à leur environnement
5. Passage de la donnée brute à l'information intelligible

Six domaines d'application stratégiques matérialisent nos recherches :
— Santé, Autonomie, Bien-être
— Ville Intelligente
— Aéronautique, Espace, Transports
— Médias Sociaux, Écosystèmes Sociaux Numériques
— E-éducation pour l'Apprentissage et l'Enseignement
— Sécurité du Patrimoine et des Personnes

Ainsi qu'**une action stratégique** :
— Calcul, masse de Données, IA

Le spectre de nos recherches est large, à la mesure des effectifs de notre unité. Il permet au laboratoire d'être un acteur incontournable des recherches, théoriques et appliquées, en Science des Données et du Calcul, qui trouvent leur application dans la vie courante, changeant nos pratiques en matière de dynamique des données, d'accès à la connaissance et d'aide à la prise de décision, au cœur du monde numérique.

D'un point de vue organisationnel, notre unité est structurée en 7 départements de recherche qui regroupent les 25 équipes du laboratoire :
— Département SI : Signaux, Images (5 équipes)
— Département GD : Gestion des Données (3 équipes)
— Département ICI : Intelligence Collective, Interaction (3 équipes)
— Département IA : Intelligence Artificielle (3 équipes)
— Département CISO : Calcul Intensif, Simulation, Optimisation (2 équipes)
— Département ASR : Architecture Systèmes Réseaux (5 équipes)
— Département FSL : Fiabilité des Systèmes et des Logiciels (4 équipes)

3 L'équipe ADRIA

Les informations de l'équipe ADRIA peuvent se trouver également sur le site web de l'IRIT :

Anciennement dénommée "Intelligence Artificielle et Robotique", puis "RPDMP" (pour "Raisonnements Plausibles, Décision et Méthodes de preuve"), l'équipe **A**rgumentation, **D**écision, **R**aisonnement, **I**ncertitude et **A**pprentissage (ADRIA) existe depuis 1976.

Elle contribue au développement de nouveaux modèles en **Intelligence Artificielle** pour le raisonnement, la décision, l'argumentation et l'apprentissage.

La représentation et le traitement des informations incomplètes, incertaines, imprécises ou incohérentes, ainsi que la représentation des préférences sont au premier plan des recherches de l'équipe ADRIA (voir le site web de l'équipe pour plus d'information).

Les **cadres formels** étudiés par l'équipe couvrent les domaines du raisonnement, de la décision, de l'argumentation et de l'apprentissage.

Les problèmes abordés utilisent des outils formels récents comme : la théorie des possibilités, et des probabilités imprécises, pour le traitement de l'incertitude épistémique ; des modèles de représentation de préférences logiques ou graphiques (VCSP, CP-nets, GAI-nets, ...) ; des logiques non-monotones, pondérées, floues, destinées à pallier les insuffisances de la logique et de l'inférence classique pour formaliser le raisonnement humain.

- **Raisonnement** : Les travaux actuels portent sur les liens entre logique possibiliste (généralisée), logiques modales, logiques multivaluées et programmation logique. La révision et la fusion d'informations incertaines et partiellement contradictoires provenant de sources multiples ainsi que le raisonnement à base de proportions analogiques sont aussi un de nos sujets d'étude actuels.
- **Décision** : L'équipe étudie les fondements axiomatiques de règles de décision qualitative sous incertitude, ainsi que des outils logiques et informatiques pour la décision multicritère et/ou collective et la planification. L'aspect combinatoire des problèmes à résoudre conduit à la recherche de classes traitables.
- **Argumentation** : L'équipe développe des modèles formels d'argumentation et des applications pour le raisonnement, l'explication de décisions, ou la modélisation de dialogues (de négociation notamment).
- **Apprentissage** : L'équipe s'intéresse à l'apprentissage artificiel (analogie, apprentissage de préférences, analyse formelle de concepts), à son adaptation dans le cadre incertain mais aussi à son interprétation en tant qu'imitation de l'apprentissage humain (apprentissage à la Piaget).

Les **travaux méthodologiques** sont le plus souvent menés en relation avec des **applications**, comme l'ordonnancement, le pilotage de chaînes logistiques, le diagnostic, l'analyse de risques, l'optimisation et la configuration. Quelques exemples d'applications :

- Critères de sécurité pour le stockage souterrain de CO2
- Système Nutri-Educ de conseil et / ou d'assistance en diététique
- Réconciliation de données pour l'analyse de flux de terres rares
- Gestion de chaînes logistiques : problèmes de planification, de coopération et de gestion des risques

Bibliographie

[1] *A060540*.

[2] D. Bigot, *Représentation et apprentissage de préférences*, PhD thesis, Mathématiques, informatique, télécommunications de Toulouse (MITT), 2015.

[3] R. Booth, Y. Chevaleyre, J. Lang, J. Mengin, and C. Sombattheera, *Learning conditionally lexicographic preference relations*, in Proceedings of the 19th European Conference on Artificial Intelligence, ECAI 2010, 2012.

[4] A. Eisa, M. Malek, and Z. Sandra, *The complexity of exact learning of acyclic conditional preference networks from swap examples*, (2018), p. 13.

[5] N. Friedman and Z. Yakhini, *On the sample complexity of learning bayesian networks*, (2013).

[6] P.-F. Gimenez, *Apprentissage de préférences en espace combinatoire et application à la recommandation en configuration interactive*, PhD thesis, Université Paul Sabatier - Toulouse III, Oct. 2018.

[7] S. Hanneke, *The optimal sample complexity of pac learning*, Journal of Machine Learning Research, 17 (2016).

[8] S.-S. Shai and B.-D. Shai, *Understanding Machine Learning : From Theory to Algorithms*, 2014.