Paramètres d'ordre et sélection de modèles en apprentissage : caractérisation des modèles et sélection d'attributs

Romaric Gaudel

N° d'ordre : 10128

Thèse

(version envoyée aux rapporteurs)
de
L'Université Paris–Sud

présentée en vue de l'obtention du
Doctorat de l'Université Paris–Sud
Spécialité : Informatique

Par
Romaric Gaudel

Équipe A&O, Laboratoire de Recherche en Informatique (LRI), U.M.R. CNRS 8623
Université Paris-Sud, 91405 Orsay Cedex, France
et
École Normale Supérieure de Cachan, antenne de Bretagne
Campus de Ker Lann, Avenue Robert Schuman, 35170 Bruz

Paramètres d'ordre et sélection de modèles en apprentissage : caractérisation des modèles et sélection d'attributs

sous la direction de Michèle Sebag et Antoine Cornuéjols

Soutenue le 14 décembre 2010 devant la commission d'examen :

M.	Stéphan Clémençon	Maître de Conférence	Examinateur
M.	Antoine Cornuéjols	Professeur	Co-directeur de thèse
M.	Luc De Raedt	Professeur	Rapporteur
M.	François Laviolette	Professeur	Rapporteur
Mme	Michèle Sebag	Directrice de Recherche	Directrice de thèse
M.	François Yvon	Professeur	Président

Remerciements

Merci à tous.

Mots clés

Apprentissage relationel, Données à instances multiples, Méthodes à noyau, Noyau-somme, Transition de phase, Sélection d'attributs, UCB appliqué aux arbres (UCT)

Résumé

Nous nous intéressons à la sélection de modèle en apprentissage automatique, sous deux angles différents.

La première partie de la thèse concerne les méthodes à noyau relationnel. Les méthodes à noyau permettent en principe de s'affranchir de la représentation des instances, et de combler le fossé entre apprentissage relationnel et apprentissage propositionnel. Cette thèse s'intéresse à la faisabilité de cet objectif dans un cas particulier : les problèmes à instances multiples, qui sont considérés comme un intermédiaire entre les problèmes propositionnels et les problèmes relationnels. Concrètement, nous déterminons sous quelles conditions le noyau-somme, utilisé sur des problèmes à instances multiples, est en mesure de reconstruire le concept-cible. Cette étude suit le schéma standard des études de transition de phase et s'appuie sur un critère nouveau pour caractériser l'efficacité de la propositionnalisation induite par le noyau-somme.

La deuxième partie de la thèse porte sur la sélection d'attributs. Une solution pour résoudre les problèmes à instances multiples, tels que présentés en première partie, passe par une propositionnalisation associant un attribut à chaque instance présente dans le problème. Le nombre d'attributs ainsi construits étant gigantesque, il est alors nécessaire de sélectionner un sous-ensemble d'attributs ne contenant que des attributs pertinents. La deuxième partie de la thèse propose donc une nouvelle approche pour la sélection d'attributs.

La sélection d'attributs est réécrite comme un problème d'apprentissage par renforcement, conduisant ainsi à une politique de sélection optimale mais non-calculable en un temps raisonnable. Cette politique est approchée en se fondant sur une approche de jeu à un joueur et en utilisant la méthode Monte-Carlo pour les arbres UCT (*Upper Confidence bound applied to Trees*), qui a été proposée par Kocsis et Szepesvári (2006). L'algorithme FUSE (*Feature Uct SElection*) étend UCT pour gérer (1) l'horizon fini mais inconnu, et (2) le facteur de branchement élevé de l'arbre de recherche reflétant la taille de l'ensemble d'attributs. Finalement, une fonction de récompense frugale est proposée en tant qu'estimation grossière mais non-biaisée de la pertinence d'un sous-ensemble d'attributs. Une preuve de concept de FUSE est fournie sur des bases de données de référence.

Title

Order parameters and model selection in Machine Learning : model characterization and feature selection

Keywords

Relational Learning, Multiple-Instance Learning, Kernel methods, Averaging kernel, Phase Transition, Feature Selection, UCB applied to trees (UCT)

Abstract

This thesis focuses on model selection in Machine Learning from two points of view.

The first part of the thesis focuses on relational kernel methods. Kernel methods hope to overcome the instances propositionalization, and to bridge the gap between relational and propositional problems. This thesis examines this objective in a particular case : the multiple instance problem, which is considered to be intermediate between relational and propositional problems. Concretely, we determine under which conditions the averaging kernel used for multiple instance problems, allows to reconstruct the target concept. This study follows the standard sketch of phase transition studies and relies on a new criterion to test the efficiency of of the propositionalization induced by the averaging kernel.

The second part of the thesis focuses on feature selection. A solution to solve multiple instance problems, as presented in the first part, is to construct a propositionalization where each instance of the problem leads to a feature. This propositionalization constructs a huge number of features, which implies the need to look for a subset of features with only relevant features. Thus, the second part of the thesis presents a new framework for feature selection.

Feature Selection is formalized as a Reinforcement Learning problem, leading to a provably optimal though intractable selection policy. This optimal policy is approximated, based on a one-player game approach and relying on the Monte-Carlo tree search UCT (*Upper Confidence bound applied to Trees*) proposed by Kocsis and Szepesvári (2006). The *Feature Uct SElection* (FUSE) algorithm extends UCT to deal with i) a finite *unknown* horizon (the target number of relevant features) ; ii) the huge branching factor of the search tree, reflecting the size of the feature set. Finally, a frugal reward function is proposed as a rough but unbiased estimate of the relevance of a feature subset. A proof of concept of FUSE is shown on benchmark data sets.

Table des matières

Chapitre 1

Introduction

La résolution des problèmes d'optimisation combinatoire et de satisfaction de contraintes comporte une transition de phase : considérons un ensemble de problèmes générés uniformément conditionnellement aux valeurs de paramètres tels que la dureté et la densité des contraintes (paramètres d'ordre). Un résultat empirique [Cheeseman 91] qui a conduit à de nombreuses études théoriques [Selman 94, Hogg 96, Mézard 05] est que la proportion de problèmes satisfiables, i.e. ayant une solution, et la complexité algorithmique de leur résolution (trouver une solution ou prouver qu'il n'en existe pas) varient brutalement autour de certaines valeurs de ces paramètres d'ordre. La région où cette variation brutale est observée, et où se trouvent les problèmes difficiles en moyenne, est appelée transition de phase. La localisation et l'étude de cette transition de phase deviennent ainsi un objectif essentiel, à la fois pour la validation de nouveaux algorithmes (benchmarking) et pour estimer *a priori* la difficulté d'un problème donné.

Une des branches de l'apprentissage automatique la plus touchée par les phénomènes de transition de phase est l'apprentissage relationnel [Giordana 00, Rückert 02, Botta 03, Pernot 05] i.e. la branche de l'apprentissage qui s'intéresse à des exemples caractérisés non pas par un n-uplet de réels (on parle alors d'apprentissage propositionnel), mais par une clef dans une base de données relationnelle. La raison pour laquelle l'apprentissage relationnel est touché par le phénomène de transition de phase est que le test de couverture relationnel (est-ce qu'une hypothèse candidate couvre un exemple) constitue un problème de satisfaction de contraintes.

Par ailleurs, l'une des approches les plus étudiées en apprentissage fait la jonction entre les deux grandes familles de problèmes (apprentissage propositionnel et apprentissage relationnel) grâce aux noyaux. L'apprentissage par noyau s'affranchit de la difficile recherche d'une représentation appropriée en se ramenant à la description d'un exemple au travers de ses relations avec les autres exemples. La question posée est de savoir si les noyaux permettent de faire disparaître toute transition de phase dans le cas des problèmes relationnels.

Notre première contribution est de montrer que la transition de phase persiste dans le cadre de l'apprentissage relationnel par noyaux. Nous avons considéré des problèmes à Instances Multiples (IM) qui sont considérés comme intermédiaires entre les problèmes propositionnels et les problèmes relationnels : pour les problèmes IM [Dietterich 97], la base de données est composée d'une seule table et un exemple est caractérisé par un ensemble de tuples de cette table connues sous le nom d'instances. La raison pour laquelle les noyaux IM considérés [Gärtner 02, Mahé 06, Kwok 07] conduisent à une transition de phase vient du fait que ces noyaux caractérisent une similarité

moyenne entre exemples alors que l'apprentissage relationnel se fixe en général pour objectif de déterminer un concept existentiel. L'approche proposée pour mettre à jour la transition de phase des noyaux IM se fonde sur un générateur aléatoire de données contrôlé par une série de paramètres d'ordre. La mise en évidence de la zone d'échec de ces noyaux s'appuie sur un critère analytique, fondé sur une nouvelle borne inférieure sur l'erreur en généralisation.

Les perspectives ouvertes par ce travail considèrent la construction d'une propositionnalisation de grande dimension, ce qui soulève dans la pratique des problèmes de sélection d'attributs [Guyon 06] : il faut sélectionner dans cette représentation les parties qui sont pertinentes pour le problème considéré. La sélection d'attributs est une problématique fondamentale en apprentissage qui a fait l'objet de nombreux travaux théoriques et algorithmiques dans la littérature [Kira 92, Tibshirani 94, Bradley 98, Fung 00, Guyon 02, Weston 03, Efron 04, Boullé 07, Chan 07, Bach 08, Shen 08, Zhang 08b, Zhang 08a, Helleputte 09, Margaritis 09, Varma 09]. La sélection d'attributs vise en particulier trois objectifs : (1) la diminution de l'erreur en généralisation en se focalisant sur les attributs pertinents ; (2) la diminution du coût d'apprentissage ou d'utilisation du modèle appris en diminuant le nombres d'attributs à traiter ; (3) l'augmentation de l'« intelligibilité » du modèle appris en fournissant un modèle plus compact.

La deuxième contribution présentée dans ce manuscrit concerne ainsi la sélection d'attributs. L'approche originale proposée est fondée sur une formalisation rigoureuse de la sélection d'attributs en terme d'apprentissage par renforcement. La solution au problème de sélection d'attributs est alors une politique optimale qui n'est pas calculable dans un temps raisonnable. Une approximation de cette politique est obtenue en se plaçant dans le cadre des jeux à un joueur. L'approche proposée s'inspire des travaux sur les méthodes Monte-Carlo pour la recherche dans un arbre (MCTS pour *Monte-Carlo Tree Search*) et de leurs avancées en particulier dans le cadre du jeu de Go [Gelly 07]. L'adaptation des méthodes MCTS au cas particulier de jeu à un joueur considéré passe (1) par la définition d'un estimateur non-biaisé et rapide à calculer de l'efficacité d'un sous-ensemble d'attributs ; (2) par une gestion de l'horizon (profondeur de l'arbre à laquelle l'algorithme s'arrête) puisque le nombre d'attributs à sélectionner est fini mais non connu à l'avance ; (3) par une politique de contrôle du facteur de branchement lors de l'exploration du treillis des sous-ensembles d'attributs. L'approche donne d'excellents résultats sur diverses bases de données de références et sur les bases de données du concours organisé à l'occasion de la conférence NIPS 2003 et portant sur la sélection d'attributs, qui est toujours utilisée comme référence de l'état de l'art.

Première partie

Transition de Phase des noyaux-sommes appliqués aux problèmes à instances multiples

L'apprentissage automatique reçoit un ensemble d'exemples et recherche le concept qui a généré ces exemples. Plus précisément, un algorithme d'apprentissage automatique choisit parmi un ensemble d'hypothèses celle qui s'approche le plus du concept-cible. L'hypothèse retournée est donc déterminée par un problème d'optimisation sous contraintes. Ce problème d'optimisation est plus ou moins difficile à résoudre suivant la représentatin des exemples, et suivant l'espace d'hypothèse considéré.

Alors que les problèmes propositionnels s'intéressent à des exemples représentés sous forme de vecteurs, les problèmes relationnels considèrent des exemples dont les informations sont stockées dans des bases de données à plusieurs tables. Les concepts manipulés dans le cadre relationnel s'écrivent donc sous forme de formules logiques du premier ordre. Pour apprendre dans le cadre relationnel on se retrouve alors confronté à une double difficulté :

- tester toutes les formules logiques est un problème combinatoire ;
- tester la validité d'une formule logique sur un exemple est un problème NP-complet ou indécidable (la littérature se ramène à un problème NP-complet par le biais de la *θ-subsomption* [Plotkin 70]).

La première difficulté est contournée en utilisant des heuristiques pour choisir les hypothèses à tester. Cela permet de restreindre le nombre d'hypothèses considérées en se concentrant sur les plus prometteuses. Même si cette modification permet d'apprendre sur de petites bases de données, elle est insuffisante pour traiter des données à grande échelle. Pour aller plus loin, une solution est de réécrire les problèmes relationnels comme des problèmes propositionnels par l'intermédiaire de noyaux relationnels [Gärtner 02, Mahé 05, Mahé 09, Mahé 06, Chen 04, Chen 06]. L'objectif est alors d'apprendre de façon efficace des concepts aussi riches qu'en manipulant directement des concepts relationnels.

Cette partie montre que l'utilisation de noyaux relationnels tels que les noyaux-somme rend impossible l'apprentissage de certains concepts relationnels. L'étude se focalise plus particulièrement sur le cas des données à instances multiples qui offrent une complexité intermédiaire entre les problèmes propositionnels et les problèmes relationnels. Le raisonnement suit la méthodologie développée pour l'étude de la transition de phase de problèmes combinatoires [Cheeseman 91, Hogg 96, Selman 94, Braunstein 05, Monasson 99, Mézard 05] et introduit une nouvelle borne inférieure sur l'erreur en généralisation.

Cette partie débute par une présentation de l'apprentissage par noyaux et de son utilisation pour l'apprentissage en présence d'instances multiples (chapitre 2). Puis le chapitre 3 met en évidence les principes des études de transition de phase ; et ces principes sont utilisés pour la définition du cadre théorique de notre étude (chapitre 4). Enfin le chapitre 5 permet de visualiser les résultats théoriques au travers de quelques expériences.

Chapitre 2

Apprentissage automatique

L'apprentissage automatique a pour objectif d'automatiser des tâches conceptuellement complexes sur des bases de données. Avec l'émergence de très grandes bases de données telles que les données clients de grandes entreprises, les données génomiques ou encore l'ensemble des résultats connus en biologie cellulaire, apparaît une difficulté : comment utiliser pleinement l'information contenue dans ces bases alors qu'il n'est plus possible pour un opérateur humain de les parcourir dans leur totalité ?

Plus formellement, l'objectif de l'apprentissage automatique est de concevoir des algorithmes capables d'apprendre un concept à partir d'exemples de ce concept. Ces exemples peuvent être fournis au départ, ou fournis un par un (apprentissage *en ligne*), ou encore choisis par l'algorithme lui même (apprentissage *actif*).

Ce chapitre se concentre sur le cas où les exemples sont fournis dès le départ. Nous présentons tout d'abord le cadre général de l'apprentissage automatique et les divers niveaux de difficulté suivant le type de données et de concepts traités (section 2.1). La section 2.2 s'intéresse plus particulièrement au paradigme des noyaux qui permet de traiter efficacement des problèmes variés en les ramenant à un problème de séparation linéaire. Puis la section 2.3 rappelle quelques-unes des applications de ce paradigme au cadre de l'apprentissage relationnel. Enfin, le chapitre conclut par une discussion sur les travaux présentés.

2.1 Plusieurs niveaux de difficulté en apprentissage

L'apprentissage automatique consiste à retrouver une hypothèse approchant au mieux un concept à partir d'exemples illustrant ce concept. Le type d'algorithmes mis en jeu ainsi que leur complexité dépendent grandement de la manière dont sont représentés les exemples et du type d'hypothèses considérées. On distingue notamment l'apprentissage propositionnel et l'apprentissage relationnel. Après avoir défini formellement l'apprentissage automatique, cette partie situe l'apprentissage propositionnel et relationnel ; precise leurs enjeux et introduit un cadre formel intermédiaire, celui de l'apprentissage à instances multiples.

2.1.1 Cadre général

Soit $\mathcal{X}$ un espace d'exemples et $\mathcal{Y}$ un ensemble d'étiquettes sur ces exemples. Un concept est une fonction $C : \mathcal{X} \to \mathcal{Y}$ associant une étiquette à chaque exemple. L'objectif d'un algorithme d'apprentissage $\mathcal{A}$ est de retrouver une hypothèse h à partir d'un ensemble d'exemples étiquetés $\mathcal{E} = \{(x_1, y_1 = C(x_1)), \ldots, (x_n, y_n = C(x_n))\}$.

En pratique, l'identification parfaite de C n'est pas toujours possible ; l'objectif devient alors de minimiser la différence entre l'hypothèse $\hat{h}$ retournée et le concept-cible C. L'écart entre le concept-cible et $\hat{h}$ est mesuré par l'erreur en généralisation

$$\mathbb{E}_x \left[L\left(\hat{h}(x), C(x)\right)\right]$$

où $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ est une fonction mesurant le coût d'une erreur de prédiction. L'objectif d'un problème d'apprentissage est donc de touver :

$$h^\star = \underset{\hat{h}\in\mathcal{X}\to\mathcal{Y}}{\operatorname{argmin}} \mathbb{E}_x \left[L\left(\hat{h}(x), C(x)\right)\right] \tag{2.1}$$

Mais l'erreur en généralisation est inconnue, un algorithme donné ne peut que l'estimer en utilisant les exemples d'entraînement. L'estimateur le plus commun est l'erreur empirique

$$\mathbb{E}_{x\in\mathcal{E}} \left[L\left(\hat{h}(x), C(x)\right)\right] = \frac{1}{n}\sum_{i=1}^{n} L\left(\hat{h}(x_i), y_i\right)$$

L'erreur empirique sous-estime l'erreur en généralisation, deux solutions sont donc adoptées pour que la minimisation de l'erreur empirique conduise à la minimisation de l'erreur en généralisation : l'ajout d'un terme de régularité et l'ajout de contraintes sur l'espace d'hypothèse (ce qui est équivalent à considérer un espace d'hypothèse $\mathcal{H} \subseteq \mathcal{X} \to \mathcal{Y}$). Ainsi les algorithmes d'apprentissage peuvent se réécrire sous la forme d'un problème d'optimisation sous contrainte :

$$\hat{h}_\mathcal{A} = \underset{\hat{h}\in\mathcal{H}}{\operatorname{argmin}} \frac{1}{n}\sum_{i=1}^{n} L\left(\hat{h}(x_i), y_i\right) + \text{régularité}\left(\hat{h}\right) \tag{2.2}$$

De la formulation de ce problème d'optimisation sous contraintes dépendent la complexité computationnelle et la qualité de l'approximation. Suivant les différents paramètres du problème ($\mathcal{X}$, $\mathcal{Y}$, $\mathcal{H}$, L, régularité, ...), la recherche de $\hat{h}_\mathcal{A}$ peut être plus ou moins coûteuse en terme de temps de calcul, ou des problèmes numériques peuvent rendre impossible le calcul exact de $\hat{h}_\mathcal{A}$.

La définition d'un algorithme d'apprentissage est donc le choix à la fois d'un espace de recherche approprié ($\mathcal{H}$ et régularité) et d'un algorithme permettant de déterminer le minimum de l'équation 2.2 ou une hypothèse approchant ce minimum. Ce choix dépend notamment de l'espace des exemples $\mathcal{X}$.

2.1.2 Apprentissage propositionnel

On parle d'apprentissage propositionnel lorsqu'un exemple est représenté par un vecteur de valeurs.

Lorsque $\mathcal{X} = \mathbb{R}^d$, des méthodes d'apprentissage « simples » sont envisageables. C'est le cas par exemple des réseaux de neurones [Bishop 95] ou des machines à vecteurs de support (section

2.2). Ces approches sont simples au sens où la recherche du minimum dans l'équation 2.2 se fait en procédant à une descente de gradient. Les calculs mis en jeu sont donc peu coûteux et il est possible de réduire le temps de calcul en s'arrêtant avant la convergence de la descente de gradient (*early stopping*), ce qui conduit à retourner une solution approchée.

Lorsque $\mathcal{X} = \{\text{vrai}, \text{faux}\}^d$, il est toujours possible de se ramener au cas $\mathcal{X} = \mathbb{R}^d$ en associant vrai à 1 et faux à 0. Mais le fait que la représentation des exemples soit booléenne conduit naturellement à introduire des espaces d'hypothèses dédiés telle que des formules en logique booléenne.

La manipulation de formules logiques introduit une difficulté en terme de temps de calcul. En effet l'espace des formules logiques est un ensemble discret et, par suite, le problème 2.2 est un problème d'optimisation combinatoire.

2.1.3 Apprentissage relationnel

L'apprentissage relationnel se confronte à tout type de données :

- des *séquences* : chaque exemple est une suite d'éléments, comme par exemple les mots d'un texte ou la séquence des nucléotides d'un gène ;
- des *arbres* : chaque exemple est un arbre. C'est le cas des données représentées en XML ;
- des *graphes* : chaque exemple est un graphe. Ces données se retrouvent notamment en biologie, où une molécule correspond au graphe des atomes la composant ;
- des *graphes* bis : chaque exemple est un nœud dans un graphe. Un exemple de tel graphe est le graphe du web, composé des pages internet et des hyper-liens les reliant entre elles ;
- des *bases de données* à plusieurs tables : de façon générale, les informations sur les produits, fournisseurs, ou clients d'une entreprise sont stockées dans des bases de données multi-relationnelles.

Les concepts à apprendre dans ces bases de données ne se limitent pas à la logique booléenne. Ces concepts s'expriment suivant les cas dans des logiques du premier ordre ou d'ordre supérieur faisant intervenir des prédicats ou des fonctions.

Dans le cadre relationnel, la recherche de la meilleure hypothèse est difficile non seulement parce que le problème d'optimisation à résoudre est un problème combinatoire, mais aussi parce que le calcul de la valeur retournée par une hypothèse sur un exemple est NP-complet ou indécidable.

Un exemple de problème relationel : Mutagenesis Mutagenesis est une base de données relationnel fondée sur les résultats de [Debnath 91], où l'objectif est de déterminer si une molécule est mutagène ou non. Pour se faire, on dispose de la description 2D des molécules. Ainsi une molécule est représentée par un ensemble d'atomes et de liens entre ces atomes. De plus, les atomes et liens sont étiquetés avec plusieurs informations telles qu'une estimation de la charge partielle ou le type de lien (liaison simple, liaison double, ...).

Une approches qui permet d'apprendre sur de telles données est la programmation logique inductive (PLI) [Srinivasan 94]. En PLI, les informations sur les atomes et liens sont représentées sous formes de prédicats tels que :

- `lien(atome1, atome2, type)` indiquant qu'il y a un lien de type `type` entre l'atome `atome1` et l'atome `atome2` ;
- `atm(mol, atome, élément, type, charge)` indiquant que la molécule `mol` contient l'atome `atome` et que cet atome est un atome de `élément` (carbone, oxygène, azote, ...), de type `type` et de charge partielle `charge`.

Il reste alors à rechercher les combinaisons de prédicats caractérisant les moécules mutagènes ou les molècules non-mutagènes.

2.1.4 Apprentissage à instances multiples

Introduit pour la première fois par Dietterich et al. [Dietterich 97], l'apprentissage à instances multiples est vu comme un intermédiaire entre l'apprentissage relationnel et l'apprentissage propositionnel.

Concept existentiel simple Originellement, un exemple est défini dans la cadre à instances multiples par un ensemble d'instances, qui sont elles-même des vecteurs. Un exemple x est positif si au moins une de ces instances vérifie un prédicat P :

$$\text{positif}(x) \Leftrightarrow \exists I \in x, P(I)$$

Ce cadre est considéré comme intermédiaire entre l'apprentissage propositionnel et l'apprentissage relationnel, puisqu'apprendre le concept dans l'espace (relationnel) des exemples revient à apprendre P dans l'espace (propositionnel) des instances. Ce problème d'apprentissage n'est tout de même pas standard, puisqu'on ne connaît pas avec certitude la classe des instances : les instances appartenant à un exemple négatif ne vérifient pas P, mais parmi les instances appartenant à un exemple positif, seules certaines, dont on ne connaît pas l'identité, vérifient P.

L'exemple de problème à instances multiples proposé dans [Dietterich 97] est celui des molécules ayant une odeur musquée. Une molécule est associée à ses différentes conformations (i.e. instances) et chaque conformation est décrite par un vecteur de dimension d. Une molécule a une odeur musquée (i.e. elle est étiquetée comme positive) si au moins l'une de ses conformations est responsable de cette odeur.

Extensions Le fait que la classe d'un exemple ne dépende que d'un seul prédicat limite les applications possibles. Ainsi [Weidmann 03] considère des concepts plus compliqués. Trois niveaux de concepts sont proposés :

1. les concepts fondés sur *la présence* : ces concepts étendent les concepts proposés dans [Dietterich 97] au cas où un exemple x nécessite de vérifier plusieurs prédicats $P_1, \ldots, P_m$ pour être considéré comme positif :
$$\text{positif}(x) \Leftrightarrow \exists I_1, \ldots, I_m \in x, \bigwedge_{i=1}^{m} P_i(I_i)$$
2. les concepts fondés sur *un seuil* : ces concepts ne considèrent plus seulement l'existence d'une instance vérifiant chaque prédicat, mais regardent pour chaque exemple x le nombre $\Delta(C_i, x)$ d'instances de x vérifiant le prédicat P_i :
$$\text{positif}(x) \Leftrightarrow \forall P_i, \Delta(P_i, x) \geq s_i$$
3. les concepts fondés sur *un dénombrement* : ces concepts vont simultanément majorer et minorer $\Delta(P_i, x)$ pour tout exemple positif x :
$$\text{positif}(x) \Leftrightarrow \forall P_i, s_i \leq \Delta(P_i, x) \leq t_i$$

Applications Un intérêt du cadre à instances multiples est la variété des applications où ce cadre intervient. En biologie moléculaire, par exemple, la représentation en deux dimensions d'une molécule peut être vue comme l'ensemble des chemins possibles dans le graphe correspondant [Gärtner 03], ou encore comme l'ensemble des sous-arbres de ce graphe [Mahé 09]. Lorsque l'on a accès à la représentation en trois dimensions d'une molécule, il est possible de considérer les zones de la molécules ayant des propriétés chimiques. Une molécule peut alors être vue comme un ensemble de triplets de telles zones actives [Mahé 06].

Le cadre à instances multiples s'applique aussi volontiers à la recherche d'images fondée sur le contenu de ces images. Ainsi une image peut être découpée en zones et chaque zone considérée comme une instance [Chen 04]. Avec cette représentation, une image de *ski* est une image contenant une zone avec de la neige et une ou plusieurs zones représentant des personnes en train de skier. Une autre représentation des images fait appel aux *régions saillantes*, i.e. qui diffèrent fortement de leur voisinage. L'idée est alors de considérer que chaque image est un ensemble de régions saillantes [Chen 06].

2.1.5 Discussion

L'apprentissage relationnel considère des bases de données et des hypothèses plus variées que l'apprentissage propositionnel. Mais cette variété àa un coût en terme de temps de calcul lors de l'apprentissage.

Une solution pour apprendre dans un temps raisonnable sur des bases de données relationnelles est de passer à une représentation propositionnelle des exemples. On parle alors de propositionnalisation [Kramer 00]. Un autre intérêt de la propositionnalisation est de séparer les deux objectifs que sont la recherche d'une bonne représentation des exemples et la recherche de la bonne hypothèse dans cette représentation.

Une autre approche pour traiter les problèmes relationnels est de les réécrire sous forme de problèmes à instances multiples [Gärtner 03, Mahé 09, Mahé 06]. On retrouve alors un problème relationnel, mais plus simple.

2.2 Les machines à vecteurs de support

Les machines a noyaux, et plus particulièrement les machines à vecteurs de support (SVM), intéressent une grande partie de la communauté de l'apprentissage automatique. Définies pour la première fois par Boser et al. [Boser 92], les machines à noyaux permettent d'étendre les propriétés statistiques des séparateurs linéaires à des séparateurs de formes variées. Ces machines se sont montrées particulièrement efficaces dans de nombreux cas réels tels que la reconnaissance de caractère [Schölkopf 95], la fouille de texte [Joachims 02] ou encore la bioinformatique [Schölkopf 04].

Positionnement On se place dans le cadre de la classification binaire : les exemples appartiennent à un espace $\mathcal{X}$ et les étiquettes sont prises dans $\mathcal{Y} = \{-1, +1\}$. On suppose que $\mathcal{X}$ est muni d'un produit scalaire noté $\langle \cdot, \cdot \rangle$. Les machines à vecteurs de support retournent une hypothèse $\hat{h} : \mathcal{X} \to \mathcal{Y}$ associant une étiquette à chaque exemple, où $\hat{h}$ est de la forme

$$\begin{aligned} \hat{h}(x) &= \text{signe}\left(\langle w, x \rangle + b\right) \\ &= \text{signe}\left(f_{w,b}(x)\right) \end{aligned}$$

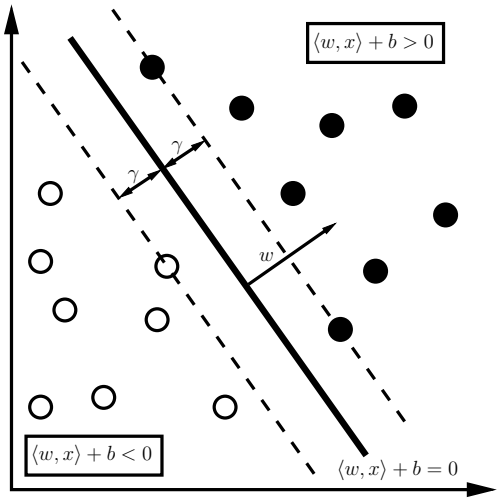


FIG. 2.1 – Les machines à vecteurs de support recherchent l'hyperplan $\langle w, x\rangle + b = 0$ qui maximise la marge $\gamma = \frac{1}{\|w\|}$. Cette hyperplan sépare les exemples positifs (ronds noirs) des exemples négatifs (ronds blancs).

avec $w \in \mathcal{X}$ et $b \in \mathbb{R}$.

Pour choisir l'hypothèse à retourner, une machine à vecteurs de support se fonde sur un ensemble d'exemples d'entraînement $\mathcal{E} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ où $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$. L'objectif est alors de déterminer parmi les couples $(w, b) \in \mathcal{X} \times \mathbb{R}$ possibles celui séparant « au mieux » les exemples d'entraînement positifs des exemples d'entraînement négatifs.

Séparateur linéaire et maximisation de la marge Les données d'entraînement sont dites linéairement séparables lorsqu'il existe un couple $(w, b) \in \mathcal{X} \times \mathbb{R}$ tel que la fonction $f_{w,b}$ associée soit positive (respectivement négative) pour les exemples d'entraînement positifs (respectivement négatifs). Dans le cas de données linéairement séparables, il existe une infinité de fonctions $f_{w,b}$ permettant de séparer les données ; parmi cette infinité de possibilités, les machines à vecteurs de support choisissent celle qui maximise la distance minimale γ entre les données et l'hyperplan $H_{w,b} = \{x \in \mathcal{X}, f_{w,b}(x) = 0\}$ (figure 2.1). Le choix de l'hyperplan maximisant la *marge* γ permet de minimiser l'erreur en généralisation puisqu'il revient à apprendre l'hypothèse la plus régulière possible ayant l'erreur empirique minimale [Vapnik 98].

Soit $H_{w,b}$ un hyperplan séparant correctement les exemples d'entraînement. Pour un exemple (x_i, y_i), sa distance à $H_{w,b}$ vaut

$$\frac{|\langle w, x_i\rangle + b|}{\|w\|}$$

où $\|\cdot\|$ est la norme associée au produit scalaire $\langle\cdot,\cdot\rangle$. Lorsque l'exemple est correctement classé par l'hyperplan, cette distance se réécrit

$$\frac{y_i\left(\langle w, x_i\rangle + b\right)}{\|w\|}$$

γ est par définition le minimum sur les exemples d'entraînement de cette distance. Mais comme multiplier w et b par une même constante ne modifie pas l'hyperplan $H_{w,b}$, on peut se limiter aux

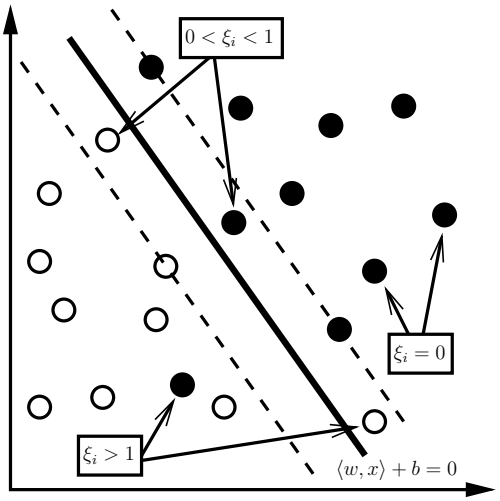


FIG. 2.2 – Machines à vecteurs de support dans le cas non-séparable : introduction des variables élastiques. Les exemples pour lesquels $\xi_i > 1$ sont mal classés ; les exemples pour lesquels $0 < \xi < 1$ sont bien classé mais dans la marge ; et les exemples pour lesquels $\xi = 0$ sont bien classé et en dehors de la marge.

hyperplans tels que

$$\min_{(x_i, y_i) \in \mathcal{E}} y_i \left(\langle w, x_i \rangle + b \right) = 1$$

On a alors

$$\gamma = \frac{1}{\|w\|}$$

Ainsi maximiser γ revient à minimiser $\|w\|$, et donc une machine à vecteurs de support retourne la solution du problème d'optimisation :

$$\underset{\substack{w \in \mathcal{X}, \\ b \in \mathbb{R}}}{\operatorname{argmin}} \frac{1}{2} \|w\|^2 \tag{2.3a}$$

$$\text{t.q. } y_i \left(\langle w, x_i \rangle + b \right) \geq 1, \qquad i = 1, \ldots, n \tag{2.3b}$$

Données non linéairement séparables Le problème d'optimisation 2.3 ne s'applique que lorsque les données sont linéairement séparables. Dans le cas contraire, il faut autoriser le fait que certaines des contraintes 2.3b ne soient pas respectées. Cela se fait en associant à chaque exemple d'entraînement x_i une *variable élastique* ξ_i positive qui permet de relaxer les contraintes 2.3b en (figure 2.2) :

$$\begin{cases} y_i \left(\langle w, x_i \rangle + b \right) \geq 1 - \xi_i, & i = 1, \ldots, n \\ \xi_i \geq 0 & i = 1, \ldots, n \end{cases}$$

ξ_i mesure l'erreur de classification de l'exemple x_i : ξ_i est plus petit lorsque l'exemple est bien classé et ξ_i vaut zéro lorsque la distance de l'exemple à l'hyperplan séparateur est plus grande que la marge. Pour minimiser l'erreur empirique il faut donc minimiser les ξ_i. La version la plus courante

des machines à vecteurs de support contrôle la somme des ξ_i :

$$\underset{\substack{w\in\mathcal{X},\\ b\in\mathbb{R},\\ \xi\in\mathbb{R}^n}}{\operatorname{argmin}} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i \tag{2.4a}$$

$$\text{t.q. } \begin{cases} y_i\left(\langle w, x_i\rangle + b\right) \geq 1 - \xi_i, & i = 1,\ldots,n \\ \xi_i \geq 0 & i = 1,\ldots,n \end{cases} \tag{2.4b}$$

La constante C sert à définir le compromis entre la maximisation de la marge et la minimisation de l'erreur empirique. Plus C est grand, plus on donne d'importance à l'erreur empirique. D'ailleurs, lorsque C tend vers plus l'infini, le problème 2.4 revient au problème 2.3.

Problème dual Généralement, la résolution du problème 2.4 se fait par l'intermédiaire de son problème dual. En introduisant les variables de Lagrange $\alpha \in \mathbb{R}^n$ et $\beta \in \mathbb{R}^n$, le minimum atteint par le problème 2.4 est égal à

$$\min_{\substack{w\in\mathcal{X},\\ b\in\mathbb{R},\\ \xi\in\mathbb{R}^n}} \max_{\substack{\alpha\in\mathbb{R}^n,\alpha\geq 0,\\ \beta\in\mathbb{R}^n,\beta\geq 0}} \mathcal{L}\left(w, b, \xi, \alpha, \beta\right) \tag{2.5}$$

où $\mathcal{L}\left(w, b, \xi, \alpha, \beta\right)$ est le lagrangien

$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i\left(y_i\left(\langle w, x_i\rangle + b\right) - 1 + \xi_i\right) - \sum_{i=1}^{n}\beta_i\xi_i$$

Le problème 2.5 est aussi équivalent à

$$\max_{\substack{\alpha\in\mathbb{R}^n,\alpha\geq 0,\\ \beta\in\mathbb{R}^n,\beta\geq 0}} \min_{\substack{w\in\mathcal{X},\\ b\in\mathbb{R},\\ \xi\in\mathbb{R}^n}} \mathcal{L}\left(w, b, \xi, \alpha, \beta\right) \tag{2.6}$$

car dans le problème initial 2.4, la fonction à minimiser 2.4a est convexe et les contraintes 2.4b sont affines.

Le minimum qui apparaît dans le problème 2.6 est atteint lorsque les dérivées partielles du laplacien $\mathcal{L}$ selon w, b, et ξ sont nulles. Ce qui conduit aux conditions :

$$w = \sum_{i=1}^{n}\alpha_i y_i x_i \tag{2.7a}$$

$$\sum_{i=1}^{n}\alpha_i y_i = 0 \tag{2.7b}$$

$$\alpha_i = C - \beta_i, \qquad i = 1,\ldots,n \tag{2.7c}$$

En injectant ces égalités dans 2.6 on obtient la version duale du problème des machines à vecteurs de support

$$\underset{\alpha\in\mathbb{R}^n}{\operatorname{argmin}} \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\alpha_i\alpha_j y_i y_j\langle x_i, x_j\rangle \tag{2.8a}$$

$$\text{t.q. } \begin{cases} \sum_{i=1}^{n}\alpha_i y_i = 0 \\ 0 \leq \alpha_i \leq C, \qquad i = 1,\ldots,n \end{cases} \tag{2.8b}$$

L'hypothèse retournée par la machine à vecteurs de support aura donc la forme

$$\hat{h}(x) = \text{signe}\left(\sum_{i=1}^{n} \alpha_i^\star y_i \langle x_i, x\rangle + b^\star\right) \tag{2.9}$$

avec $\alpha^\star$ la solution du problème 2.8 et $b^\star$ calculé en utilisant les conditions de *Karush-Kuhn-Tucker* :

$$\alpha_i^\star \left(y_i \left(\langle w^\star, x_i\rangle + b^\star\right) - 1 + \xi_i^\star\right) = 0, \qquad i = 1, \ldots, n \tag{2.10a}$$

$$\beta_i^\star \xi_i = 0, \qquad i = 1, \ldots, n \tag{2.10b}$$

Pour un exemple (x_i, y_i) tel que $\alpha_i^\star$ est strictement compris entre 0 et C, ces conditions impliquent que $b^\star$ vaut

$$y_i - \langle w^\star, x_i\rangle$$

L'astuce du noyau Globalement, que ce soit au cours de la résolution du problème 2.8 ou de l'utilisation de l'hypothèse obtenue, on ne fait jamais appel à un exemple directement. Seuls des produits scalaires entre exemples sont utilisés. Il est donc possible de remplacer ce produit scalaire par n'importe quel *noyau*, i.e. n'importe quelle fonction $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ vérifiant les conditions de Mercer :

1. k est symétrique ;
2. pour tout ensemble de points $x_1, \ldots, x_n$, la matrice de noyau associée K, définie par $K_{i,j} = k(x_i, x_j)$, est semi-definie positive.

Les conditions de Mercer impliquent que k correspond à un produit scalaire dans un espace particulier $\mathcal{Z}$, i.e. qu'il existe une fonction $\Phi : \mathcal{X} \to \mathcal{Z}$ telle que pour tout couple d'exemples (x, x'),

$$k(x, x') = \langle \Phi(x), \Phi(x')\rangle_{\mathcal{Z}}$$

Dans le cas où les exemples sont des vecteurs de réels ($\mathcal{X} = \mathbb{R}^d$), un intérêt des noyaux est de pouvoir rendre linéairement séparables dans $\mathcal{Z}$ des données qui ne le sont pas dans $\mathbb{R}^d$ en utilisant par exemple un noyau *polynomial* ($k(x, x') = (\langle x, x'\rangle + 1)^a$) ou un noyau *gaussien* ($k(x, x') = \exp\left(-\frac{\|x-x'\|^2}{2\sigma^2}\right)$).

Mais, de façon plus générale, l'astuce du noyau permet d'utiliser le cadre des machines à vecteurs de support sur toute sorte de données dès lors qu'il est possible de définir un noyau. [Joachims 02] et [Schölkopf 04] présentent, par exemple, des noyaux adaptés respectivement à la fouille de texte et à la bioinformatique.

Pour finir, l'hypothèse retournée, telle que décrite par l'équation 2.9, fait intervenir tout les exemples d'entraînement. L'espace mémoire utilisé pour le stockage de $\hat{h}$, ainsi que le temps de calcul nécessaire à l'évaluation de $\hat{h}$ sur un exemple, augmentent donc linéairement avec le nombre d'exemples d'entraînement. Cette augmentation limite les possibilités de passage à l'échelle pour l'approche.

Mais n'interviennent dans l'hypothèse retournée que les *vecteurs de support*, i.e. les exemples dont la variable de Lagrange α_i est non-nulle. Tant que le nombre de ces vecteurs reste modéré, les machines à vecteurs de support à base de noyaux restent utilisables.

2.3 Quelques noyaux relationnels

Comme indiqué en section 2.1, la taille de l'espace d'hypothèse explose dans le cadre de l'apprentissage relationnel. Cette explosion induit un coût de calcul élevé pour trouver la meilleure hypothèse. Ce coût est d'autant plus élevé que la recherche de la meilleure hypothèse passe par la résolution d'un problème combinatoire.

A contrario, les machines à vecteurs de support (section 2.2) fournissent un cadre pour apprendre « facilement » une hypothèse à partir de n'importe quel type de données. L'hypothèse est apprise facilement puisqu'elle est définie par le biais d'un problème quadratique sous contraintes linéaires (équation 2.8).

De plus, le cadre des machines à vecteurs de support est applicable à n'importe quel type de données grâce à l'astuce des noyaux. Que ce soit lors de l'apprentissage ou lors de l'utilisation de l'hypothèse, il n'est jamais nécessaire de fournir la représentation des exemples. La seule information utilisée est la similarité entre deux exemples, mesurée à partir d'un noyau k.

Les machines à vecteurs de support sont donc un candidat intéressant pour se confronter aux problèmes relationnels sans pour autant payer un coût de calcul trop élevé. Toute la problématique est de définir un noyau sur les données relationnelles qui soit à la fois calculable en un temps raisonnable et permette également l'apprentissage d'hypothèses suffisamment riches.

Cette section fournit quelques exemples de noyaux utilisés pour traiter des données relationnelles.

2.3.1 Noyaux pour données textuelles

Le traitement de données textuelles apparaît dans de nombreux contextes : la séparation de news en fonction de leur thématique, le filtrage de mails, ou la recherche de pages internet. Un texte est en soi une suite de caractères ou de mots ; on est donc dans le cadre de l'apprentissage relationnel.

Une approche courante consiste à oublier le coté séquentiel des données et à considérer les textes comme un *sac de mots* (bag of words) [Joachims 98]. Plus précisément, un texte est représenté par un ensemble d'attributs. Chaque attribut est associé à un mot (ou à une racine de mots) et sa valeur correspond au nombre d'occurrences de ce mot. On se retrouve donc dans le cadre propositionnel.

Cette propositionnalisation d'un texte oublie l'information fournie par l'ordre des mots. Pour ne pas perdre cette information, [Lodhi 02] propose un noyau K_m travaillant directement sur les séquences de caractères.

Ce noyau est défini à partir de son espace de représentation sur les textes. Pour un texte t donné, à chaque m-uplet de caractère est associé un décompte du nombre de fois où ce m-uplet est présent dans le texte. Plus précisément, on regarde toutes les façons possibles de considérer ce m-uplet comme une sous-chaîne de caractère de t (potentiellement discontinue dans t), et à chaque positionnement est associé le score λ^i, où i est la distance entre le positionnement du dernier et du premier caractère du m-uplet et λ un paramètre plus petit que 1.

Un calcul de ce noyau par le biais de sa représentation coûterait $\mathcal{O}(|\Sigma|^m)$ en temps et en espace, où Σ est l'alphabet. Mais [Lodhi 02] utilise la programmation dynamique pour fournir une méthode calculant $K_m(s,t)$ en $\mathcal{O}(m|s||t|)$. La complexité devient alors acceptable pour de petites bases de données mais trop élevée pour être applicable sur de grands corpus.

Pour permettre le passage à l'échelle de son approche, [Lodhi 02] propose une version approchée du calcul de K_m. Cette version revient à n'utiliser que certains des attributs de la représentation associée à K_m.

Les expériences proposées dans [Lodhi 02] montrent que sur de petites bases de données (moins

de 500 exemples), le noyau K_m permet d'obtenir de meilleur résultats que les sacs de mots. Sur des bases de données plus importantes (plusieurs milliers d'exemples), les deux approches sont équivalentes. Le noyau fondé sur les sous-chaînes de caractères est donc capable de détecter des informations subtiles que l'approche à base de sacs de mots ne peut reconstruire qu'à l'aide d'un grand nombre d'exemples.

2.3.2 Noyau-somme

Le cadre des données à instances multiples considère des exemples définis comme des ensembles d'instances (2.1.4). Selon la définition première de ce cadre, un exemple est positif si une de ses instances vérifie une propriété P [Dietterich 97]. Ce problème est de type relationnel, mais si on se situe au niveau des instances, on retrouve un problème propositionnel. En effet, chaque instance est représentée par un vecteur, et dans l'espace des instances, l'objectif est d'apprendre P à partir d'exemples négatifs de P (les instances d'exemples négatifs) et d'exemples potentiellement positifs (les instances des exemples positifs).

[Gärtner 02] propose un noyau adapté aux problèmes à instances multiples : *le noyau-somme*. Ce noyau considère tous les couples d'instances entre une instance du premier exemple et une instance du deuxième exemple, calcule la valeur prise par un noyau k sur ces couples d'instances, met à la puissance a les valeurs mesurées, somme toutes les valeurs obtenues, et enfin normalise la somme. Formellement, le noyau-somme K pour deux exemples x et x' est défini par

$$K(x,x') = \frac{\sum_{x_i \in x} \sum_{x_j \in x'} k(x_i,x_j)^a}{f_{\text{norm}}(x) f_{\text{norm}}(x')} \tag{2.11}$$

avec f_{norm} une fonction de normalisation.

[Gärtner 02] montre que si P est apprenable dans l'espace des instance à l'aide du noyau k, alors pour a suffisamment grand, le concept à instances multiples faisant intervenir P est apprenable à l'aide du noyau K.

[Gärtner 02] remarque que pour les noyaux d'instance courants, choisir a revient à modifier les paramètres du noyau. Nous nous limiterons donc dans la thèse au cas où $a = 1$. De plus, les expériences menées dans [Gärtner 02] indiquent que les meilleurs résultats sont obtenus en choisissant comme fonction de normalisation soit $\sqrt{K(x,x)}$, soit $|x|$ (nombre d'instances dans x). C'est pourquoi, au final, le noyau que nous considérons dans cette thèse est le suivant :

$$K(x,x') = \frac{1}{|x|}\frac{1}{|x'|} \sum_{x_i \in x} \sum_{x_j \in x'} k(x_i,x_j) \tag{2.12}$$

Applications Le noyau-somme a été testé sur de nombreuses bases de données. Dans [Gärtner 02], le noyau-somme est appliqué à deux bases de données. La première base de données (MUSK), qui est la base de données prototype des problèmes à instances multiples (section 2.1.4), regroupe des molécules qui ont ou non une odeur musquée. Chaque molécule est représentée par un ensemble de conformations, et chaque conformation est associée à un vecteur d'attributs. La deuxième base de données (Mutagenesis) s'intéresse elle aussi à des molécules, mais représentées cette fois-ci par leur graphe. Plus précisément, chaque molécule est associée à l'ensemble de ses liens entre deux atomes.

Mahé a également utilisé le noyau-somme pour classer des molécules. Dans [Mahé 05], chaque molécule est associée aux chemins dans son graphe. Soit $G = (V_G, E_G)$ un graphe composé des sommets V_G et des arêtes E_G. On note V_G^* l'ensemble des chemins de taille finie dans ce graphe.

Soit $l : V_G \cup E_G \to \mathcal{A}$ une fonction étiquetant les sommets et les arêtes de G dans un alphabet fini $\mathcal{A}$. On note $l : V_G^* \to \mathcal{A}^*$ la fonction qui à tout chemin $w = v_1 v_2 \dots v_k$ associe le mot correspondant à la liste des étiquettes rencontrées en parcourant w

$$l(w) = l(v_1)l(v_1, v_2)l(v_2)\dots l(v_k)$$

Le noyau-somme étudié par Mahé prend la forme

$$K(G_1, G_2) = \sum_{w_1 \in V_{G_1}^*} \sum_{w_2 \in V_{G_2}^*} p_{G_1}(w_1) p_{G_2}(w_2) k(l(w_1), l(w_2))$$

où p_G est une distribution de probabilité sur les chemins de G et k un noyau sur les mots de $\mathcal{A}^*$.

Par la suite, [Mahé 09] étend l'idée des chemins d'un graphe aux sous-arbres recouvrants. Enfin [Mahé 06] étudie la représentation d'une molécule obtenue en considérant tous les triplets de « parties actives » dans la molécule.

Mais le noyau-somme donne aussi des résultats intéressants pour la classification d'images, où une image peut être vue comme un ensemble de zones [Chen 04] ou bien un ensemble de points saillants [Chen 06].

Discussion [Gärtner 02] démontre qu'un problème à instances multiples est apprenables par le noyau-somme à condition que la propriété P sur les instances associé à ce problème soit elle même apprenable par le noyau sur les instances. De plus le noyau-somme donne de bons résultats empirique sur plusieurs problèmes à instances multiples.

Or [Weidmann 03] définit des concepts à instances multiples plus compliqués faisant intervenir plusieurs propriétés au lieu d'une seule. Pour certaines applications, ces concepts semblent plus adaptés. Par exemple, dans le cas de la classification d'images représentées par des zones, on s'attend à ce qu'une photo d'un « coucher de soleil sur la mer » contienne une zone représentant le soleil et une zone représentant la mer.

Une question ouverte à notre connaissance est de savoir si les concepts proposés par [Weidmann 03] sont eux aussi apprenables à l'aide du noyau-somme. Le chapitre 4 étudie donc les conditions sous lesquelles le noyau-somme permet d'apprendre ou non des concepts faisant intervenir plusieurs propriétés.

2.4 Résumé

L'apprentissage automatique se fixe pour objectif de retrouver un concept à partir d'exemples de ce concept. On se place dans le cas particulier où les exemples sont de la forme (x_i, y_i) avec $x_i \in \mathcal{X}$ l'exemple à proprement parler, et $y_i \in \mathcal{Y}$ l'étiquette associée à l'exemple. L'objectif est alors de déterminer la fonction $C : \mathcal{X} \to \mathcal{Y}$ qui associe une étiquette à chaque exemple.

La recherche de C, ou de la fonction $\hat{h}$ s'en approchant le plus, passe le plus souvent par la résolution d'un problème d'optimisation sous contraintes. Ce problème traduit les deux objectifs sous-jacents à l'apprentissage :

1. trouver une fonction $\hat{h}$ qui ait la plus petite erreur possible sur les exemples d'entraînement ;
2. trouver une fonction $\hat{h}$ qui soit la plus simple possible.

Ces deux objectifs permettent de trouver la fonction $\hat{h}$ la plus proche possible de C, i.e. la fonction $\hat{h}$ dont le comportement sur d'autres exemples que les exemples d'entraînement sera le plus proche possible de celui de C.

Suivant l'espace $\mathcal{X}$ dans lequel sont définis les exemples et suivant l'ensemble d'hypothèses dans lequel est recherchée $\hat{h}$, le problème d'optimisation est plus ou moins difficile à résoudre. Le cadre le plus difficile est le cadre relationnel, où les informations sur les exemples sont stockées dans une base de données relationnelle. Dans ce cadre, une hypothèse est une formule logique. La recherche de $\hat{h}$ est alors extrêmement coûteuse puisque le problème d'optimisation recherchant la meilleure hypothèse ainsi que le problème correspondant au test de $\hat{h}$ sur un exemple sont tous les deux un problème NP-complet.

Dans le cadre plus simple de l'apprentissage propositionnel, un exemple est exprimé sous forme d'un vecteur (un enregistrement unique d'une base de données). La situation « la plus simple » est lorsque les exemples sont représentés par un vecteur de réels ($\mathcal{X} = \mathbb{R}^d$) et que les étiquettes sont binaires ($\mathcal{Y} = \{-1, 1\}$). Il est alors possible de rechercher un hyperplan séparant les exemples négatifs des exemples positifs. Les machines à vecteurs de support, par exemple, recherchent un tel hyperplan par le biais de la résolution d'un problème quadratique sous contraintes linéaires. Un tel problème a l'avantage de pouvoir se résoudre en un temps raisonnable grâce à une simple descente de gradient.

Afin de pouvoir apprendre sur des données relationnelles en un temps raisonnable, une solution est de se ramener au cadre propositionnel. Une première approche pour obtenir une version propositionnelle du problème est de construire une liste de descripteurs sur les exemples. On parle alors de propositionnalisation [Kramer 00]. La problématique de la recherche de la bonne hypothèse dans l'espace relationnel est alors découpée en deux sous-problématiques plus simples : (1) la recherche de la bonne représentation, (2) la recherche de la bonne hypothèse dans l'espace de représentation.

Une autre approche pour revenir au cadre propositionnel est de ne pas étudier les exemples directement, mais de s'intéresser uniquement aux liens qu'ils ont entre eux. Cette approche prend tout son sens dans la cadre des machines à vecteurs de support. En effet, lors de l'apprentissage et lors de l'utilisation de l'hypothèse apprise, ces machine ne font appel qu'au calcul d'un *noyau* k entre deux exemples. Ces machines, faciles à apprendre, peuvent donc s'utiliser sur des données relationnelles, à condition de fournir un noyau entre deux exemples relationnels qui soit calculable dans un temps raisonnable, tout en fournissant une information suffisante pour reconstruire des concepts compliqués.

Les méthodes de propositionnalisation, ou l'utilisation d'un noyau, permettent de traiter les problème relationnels comme des problèmes propositionnels. Mais cette réécriture se fait-elle aux dépends de la richesse des concepts apprenables ? Y-a-t-il des concepts que l'on ne peut pas reconstruire une fois que l'on est passé dans le cadre propositionnel ?

La première partie de cette thèse examine ces questions au travers d'un cas particulier : le noyau-somme utilisé dans le cadre des exemples à instances multiples. Cette étude suit le schéma des études sur la transition de phase. Ce schéma est présenté dans le chapitre suivant.

Chapitre 3

Transition de Phase et cartes de compétence

La notion de transition de phase a été introduite en physique statistique. Elle correspond au changement d'état très rapide lorsque l'on fait varier certains paramètres autour d'une valeur critique. Cette notion a par la suite été reprise en informatique où ce changement d'état a été mis en évidence dans le cas de la résolution de problèmes NP-complets. Dans le cadre de l'informatique, les recherches sur la transition de phase ont permis d'identifier les problèmes réellement difficiles [Cheeseman 91, Hogg 96], et de proposer des algorithmes adaptés à la résolution de ces problèmes [Selman 94, Braunstein 05]. Enfin, certaines études permettent de caractériser l'espace des solutions des problèmes NP afin de mieux comprendre l'origine de la transition de phase [Monasson 99, Mézard 05].

L'apprentissage automatique fait parfois lui-même appel à des problèmes d'optimisation combinatoire (section 2.1. Plusieurs travaux se sont donc intéressés à l'impact des phénomènes de transition de phase en apprentissage. On retrouve notamment des travaux en programmation logique inductive (ILP) [Giordana 00], en apprentissage propositionnel [Rückert 02], et en inférence grammaticale [Pernot 05]. Comme pour le cadre des problèmes NP-complets, ces recherches ont permis de caractériser les problèmes difficiles [Botta 03, Baskiotis 04, Pernot 05] et d'élaborer des algorithmes performants pour ces problèmes [Bianchetti 02, Rückert 03, Maloberti 04].

Ce chapitre introduit brièvement les travaux sur la transition de phase (section 3.1) et discute des effets de la transition de phase sur un cadre d'apprentissage particulier (section 3.2). Enfin la section 3.3 résume le fonctionnement et les apports des études sur la transition de phase.

3.1 Transition de Phase des problèmes de satisfaction de contraintes

Les problèmes *NP-complets* sont les problèmes solvables en un temps polynomial par une machine de Turing non-déterministe. Ce sont donc des problèmes pour lesquels il est possible de tester une solution en temps polynomial. Mais lorsque le nombre de solutions possibles est exponentiel, il devient très long pour un ordinateur standard de vérifier si le problème est satisfiable ou non, puisqu'un tel ordinateur ne peut tester qu'un nombre fini de solutions à la fois.

La plupart des problèmes de satisfaction de contraintes sont NP-complets. C'est le cas de nombreux problèmes industriels tels que la création d'emploi du temps ou la planification de production. Leur résolution est donc au pire cas longue, mais en moyenne les algorithmes existants trouvent le plus souvent une solution en un temps raisonnable. De cette remarque découle la volonté de caractériser les problèmes faciles [Karp 83, Purdom 83].

Cette volonté a abouti à plusieurs études sur différents types de problèmes NP-complets [Zhang 04, Xu 05, Cocco 01]. La conclusion de ces études est toujours la même : pour certains paramètres d'ordre, les problèmes passent brutalement d'une situation ouù ils sont presque tous satisfiables à une situation où ils sont presque tous non-satisfiables, et les problèmes difficiles se trouvent en moyenne au niveau de ce changement brutal. Cette transition rapide d'un état à un autre est couramment appelée *transition de phase* par analogie avec les transitions de phase observées en physique statistique.

Cette section présente les résultats obtenus pour le premier problème défini comme NP-complet : le problème 3-SAT [Cook 71]. L'objectif est de mettre en évidence la démarche suivie dans les études sur la transition de phase et de préciser le type de résultats attendus de la part de ces études.

3.1.1 Observations empiriques

Le problème 3-SAT a pour objectif de satisfaire un ensemble de contraintes exprimées sous la forme de 3-CNF.

Formellement, une *clause* est la disjonction de variables booléennes, chaque variable étant donnée sous forme positive ou négative. On appelle *forme normale conjonctive* (CNF) une conjonction de clauses. On parle de k-CNF lorsque ces clauses comportent moins de k variables chacune. Le problème 3-SAT consiste donc à déterminer pour une formule 3-CNF F donnée s'il existe une instantiation de ses variables telle que F soit vrai. Si une telle instantiation existe, la formule est dite *satisfiable*.

Identification de la transition de phase : modèle de données paramétré [Mitchell 92] a mis en évidence un phénomène de transition de phase sur les problèmes 3-SAT. Pour ce faire, des problèmes 3-SAT sont générés aléatoirement selon des paramètres dits *paramètres d'ordre*, puis on observe comment évolue la proportion de problèmes satisfiables suivant ces paramètres d'ordre.

Deux paramètres régissent la génération de problèmes 3-SAT : le nombre d de variables et le nombre n de clauses. Ainsi chacune des n clauses d'un problème 3-SAT est obtenue en choisissant uniformément trois variables parmi les d variables existantes. Ces variables sont utilisées de façon positive ou négative avec probabilité 1/2.

On observe alors comment évolue la probabilité pour les problèmes générés d'être satisfiables en fonction du paramètre $\kappa = \frac{n}{d}$. On remarque que les problèmes générés passent d'un état où ils sont presque tous satisfiables à un état où ils sont presque tous non-satisfiables. Indépendamment du nombre de variables utilisées, ce passage est brutal et autour d'une valeur critique $\kappa_c \approx 4.27$. On a donc bien un phénomène que l'on peut considérer comme une transition de phase.

Si l'on s'intéresse au contraire au temps de calcul du solveur de contraintes, il admet un pic autour de κ_c. Ainsi la zone de transition correspond à une zone où les problèmes 3-SAT sont difficiles à résoudre. Cette remarque corrobore l'intuition :

- pour κ petit devant κ_c, chaque problème est peu contraint et la plupart des instantiations de variables permettent de valider le problème. L'algorithme trouve donc rapidement une solution ;

- pour κ grand devant κ_c, chaque problème est fortement contraint et il suffit d'instancier peu de variables pour se rendre compte qu'elles sont incompatibles. L'algorithme démontre donc rapidement qu'il n'existe pas de solution ;
- pour κ proche de κ_c, soit le problème est vrai pour peu d'instantiations (qui seront donc difficiles à trouver), soit le problème n'admet pas de solutions, mais il faut instancier la quasi totalité des variables pour se rendre compte qu'une instantiation ne valide pas le problème.

Conséquences de la transition de phase Mettre en évidence une transition de phase permet tout d'abord d'identifier les problèmes réellement difficiles. Ainsi les algorithmes peuvent être comparés dans cette zone critique afin de connaître au mieux leur capacité.

Cette découverte a aussi encouragé des recherches analytiques (section suivante) dont ont découlé des algorithmes dédiés et donc efficaces même lorsque l'on est proche de κ_c [Dubois 01, Wei 04, Braunstein 05].

3.1.2 Études analytiques

Ces résultats expérimentaux intéressent la communauté de la physique statistique. Cette communauté dispose d'outils aptes à étudier analytiquement le comportement « macroscopique » de systèmes mettant en relation un grand nombre d'éléments suivant une loi « microscopique » simple. Les premières études analytiques fondées sur ces outils ont, par exemple, permis de fournir des bornes théoriques sur la valeur de κ_c [Dubois 00, Achlioptas 01, Mézard 02].

D'autres études tentent d'expliquer le comportement des solutions d'un problème de contraintes en fonction de κ. Certaines s'intéressent par exemple à la structure des ensembles de solutions [Mézard 05]. Lorsque κ est petit, toutes les solutions se retrouvent dans une même clique. Puis lorsque κ augmente, cette clique explose en plusieurs cliques dont le nombre s'accroît avec κ. Enfin, lorsque κ est proche de κ_c, le nombre de cliques augmente exponentiellement [Achlioptas 06].

3.2 Transition de Phase en apprentissage automatique : un exemple

L'apprentissage automatique fait appel le plus souvent à un problème d'optimisation sous contraintes (section 2.1). Dans le cas particulier de l'apprentissage relationnel, c'est un problème d'optimisation combinatoire NP-complet qui est considéré. Or de nombreux problèmes NP-complets engendrent une transition de phase (section 3.1). Quelques études sont donc consacrées à la recherche de transitions de phases en apprentissage automatique et aux conséquences de ces transitions [Giordana 00, Botta 03, Rückert 02, Rückert 03, Baskiotis 04, Hollanders 07, Alphonse 08a, Alphonse 08b, Saitta 10].

Cette section se focalise sur un cadre d'apprentissage relationnel particulier : la programation logique inductive (ILP). Tout d'abord cette section définit le cadre d'apprentissage de l'ILP, puis met à jour une transition de phase pour l'un des outils de l'ILP : la θ-subsomption [Plotkin 70]. Enfin, cette section s'intéresse aux conséquences de cette transition de phase sur la qualité des hypothèses apprises.

3.2.1 ILP

L'ILP manipule des données et hypothèses décrites en logique du premier ordre. Une hypothèse fait donc intervenir des variables, des fonctions et des prédicats. En particulier, les constantes sont

représentées par des fonctions d'arité zéro, et les prédicats sont à valeurs booléennes.

Un exemple x est considéré comme positif d'après une hypothèse h et un ensemble de connaissances b si x est conséquence logique de h et b ($h, b \vDash x$). Au contraire, l'exemple est considéré comme négatif s'il n'est pas conséquence logique de h et b ($h, b \nvDash x$). Mais la conséquence logique est indécidable, elle est donc remplacée par la θ-subsomption, qui est un problème NP-complet [Plotkin 70].

Formellement, le langage manipulé se définit comme suit.

- *terme* : un terme se définit de façon récursive. Les variables et les symboles de fonctions d'arité zéro (les constantes) sont des termes. Si f est une fonction d'arité n et si $t_1, \ldots, t_n$ sont des termes, alors $f(t_1, \ldots, t_n)$ est un terme.
- *formule atomique, littéral, littéral positif, littéral négatif* : Si R est un prédicat d'arité n et $t_1, \ldots, t_n$ sont des termes, alors $R(t_1, \ldots, t_n)$ est une formule atomique.
  Un littéral est une formule atomique (littéral positif) ou sa négation (littéral négatif).
- *clause* : une clause est une formule logique de la forme $\forall x_1, \ldots, x_n \bigvee_{i=1}^{m} L_i$ où $x_1, \ldots, x_n$ sont les variables apparaissant dans les littéraux $L_1, \ldots, L_m$.
- *clause de Horn* : Une clause de Horn est une clause contenant au moins un littéral positif.
- *clause de programme/définie, tête de programme et corps de programme* : une clause de programme (ou clause définie) est une clause comprenant exactement un littéral positif h. Elle peut donc s'écrire sous la forme $h \leftarrow L_1, \ldots, L_m$.
  h est appelé la tête du programme et $L_1, \ldots, L_m$ son corps.
- *fait* : Un fait est une clause de programme dont le corps est vide.
- *substitution* : Soit $x_1, \ldots, x_n$ des variables et $t_1, \ldots t_n$ des termes. La substitution $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$ est une fonction qui à toute clause C associe la clause $C' = C\theta$ où les occurrences de $x_1, \ldots, x_n$ dans C ont été remplacées respectivement par $t_1, \ldots t_n$.
- *θ-subsomption, test de θ-subsomption* : Soit C et C' deux clauses. C θ-subsume C', s'il existe une substitution θ telle que $C\theta \subseteq C'$, i.e. tous les littéraux de $C\theta$ appartiennent à C'. Le test de θ-subsomption entre deux clauses C et C' est le test qui retourne vrai si C θ-subsume C'.

Dans le cadre de l'ILP, les exemples sont des faits ou des classes instanciées [Landwehr 07] et les hypothèses apprises sont des programmes logiques exprimés à l'aide d'ensembles de clauses de programme. L'objectif est de trouver un programme logique qui couvre (au sens de la θ-subsomption) les exemples positifs et pas les exemples négatifs.

3.2.2 Transition de phase de la θ-subsomption

[Giordana 00] s'intéresse au test de θ-subsomption, qui est utilisé en interne dans le cadre de l'ILP. L'étude montre l'existence d'une transition de phase pour ce test.

Génération des données Plus précisément, l'étude s'intéresse au test de θ-subsomption entre une clause C et un exemple E représenté par un ensemble de tables relationnelles. Afin que le test ne puisse pas être découpé en sous-problèmes, la clause C est choisie connectée, et chaque prédicat n'apparaît qu'une seule fois par clause. Par simplicité et sans pert de généralité, les prédicats sont d'arité 2 et sur un même domaine de constantes $\mathcal{X}$. Enfin, un exemple ne contient que des prédicats « utiles », i.e. des prédicats apparaissant dans C.

Quatre paramètres d'ordres caractérisant les problèmes (C, E) générés :

- le nombre m de prédicats par clause ;
- le nombre n de variables par clause ($m < n$) ;

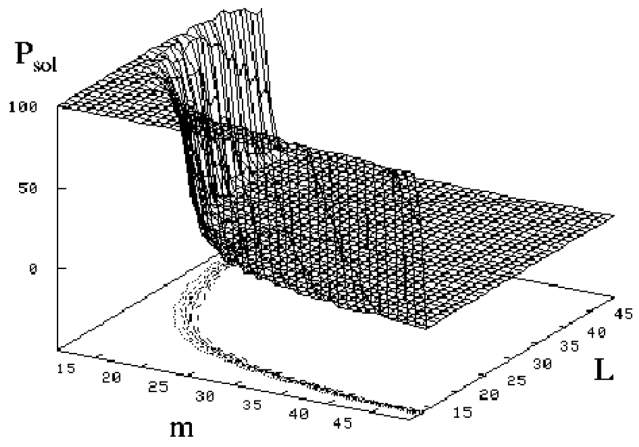


FIG. 3.1 – Proportion de tests de θ-subsomption positifs en fonction du nombre L de constantes et du nombre m de prédicats par clause.

- la taille N des tables de prédicats (pour un prédicat R donné, $R(c_1, c_2)$ est vrai pour exactement N couples de constantes (c_1, c_2)) ;
- le nombre L de constantes.

Résultats empiriques Pour chaque quadruplet (n, N, m, L), cent problèmes sont générés aléatoirement et résolus à l'aide d'un algorithme exhaustif. Puis on affiche la proportion de tests de θ-subsomption positifs, dans le plan (m, L) (figure 3.1). Quelles que soient les valeurs des paramètres m et N, on observe alors une transition de phase. Pour m ou L petits, C θ-subsume toujours E. On parle donc de *région « YES »*. Pour m et L grands, au contraire, C ne θ-subsume jamais E. Cette région est donc nommée *région « NO »*. Enfin, la transition entre ces deux régions est très abrupte.

On peut aussi s'intéresser à la complexité computationnelle (exprimée en nombre d'itérations avant de trouver une substitution ou de prouver la non-existence d'une substitution). Comme pour l'étude des problèmes 3-SAT (section 3.1), cette complexité admet un pic au niveau de la transition de phase.

3.2.3 Transition de phase de l'apprentissage

Une fois la transition de phase du test de la θ-subsomption mise en évidence, il reste à déterminer son impact sur l'apprentissage.

Un première difficulté provient non pas directement de la transition de phase, mais de l'existence de plateaux (les régions YES et NO). Les algorithmes de la littérature, notamment les arbres de décision [Breiman 84] ou les approches fondées sur la clause plancher [Muggleton 95], débutent leur recherche dans ces zones. Comme ce sont des plateaux, il leur sera impossible de guider leur recherche [Pearl 84], qu'ils soient fondés sur le *gain en information* [Quinlan 90] ou un critère de *longueur de description minimale* [Rissanen 78, Muggleton 95].

Données réelles Une deuxième difficulté est visible sur les données réelles : les hypothèses apprises appartiennent à la région de la transition de phase et induisent un coût de calcul important. Plus précisément, non seulement l'hypothèse apprise est dans la transition de phase, mais les hypothèses testées au cours de l'apprentissage s'en sont rapprochées petit à petit. Or le test de chaque hypothèse nécessite plusieurs calculs de θ-subsomption, qui deviennent donc très coûteux à mesure que l'on se rapproche de la zone de transition.

[Giordana 00] observe cette propriété sur deux bases de données réelles. Cette observation peut s'expliquer par l'affirmation que la plupart des problèmes de la vie courante sont sur la zone de transition, puisque cette zone est par définition celle à laquelle appartiennent les concepts permettant de discriminer les exemples.

Mais le fait que les hypothèses apprises sur des données réelles soient dans la zone de transition de phase peut aussi s'expliquer par un biais durant l'apprentissage. L'algorithme d'apprentissage recherche une hypothèse qui couvre les exemples positifs et pas les exemples négatifs. Or les hypothèses de la région YES (respectivement NO) couvrent la majorité des exemples (respectivement couvrent peu d'exemples). Il est donc difficile de trouver une hypothèse acceptable dans la région YES ou la région NO. Dans la zone de transition de phase, au contraire, il est plus facile de trouver une hypothèse discriminant correctement les exemples d'entraînement.

[Giordana 00] met en évidence ce comportement en recherchant la probabilité de discriminer deux exemples (ce qui correspond au cas où on a exactement un exemple négatif et un exemple positif dans la base d'entraînement). On remarque alors que la probabilité de générer une hypothèse discriminant ces deux exemples est quasiment nulle, excepté près de la transition de phase. Cette probabilité atteint même 50% au niveau de la transition.

Données artificielles [Botta 03] met en évidence deux autres difficultés découlant de l'existence d'une transition de phase pour la θ-subsomption :

- lorsque l'hypothèse à apprendre est sur la transition de phase, ou dans la région NO proche de la transition de phase, les algorithmes testés sur-apprennent ;
- lorsque l'hypothèse à apprendre est dans la région NO loin de la transition de phase, les algorithmes testés apprennent une sur-généralisation du concept cherché.

Plus précisément, [Botta 03] génère des problèmes d'apprentissage dans les différentes régions et apprend une hypothèse. Le fait de connaître le problème généré permet de vérifier si l'hypothèse apprise est bien celle recherchée. L'hypothèse est apprise à partir de FOIL [Quinlan 90], qui construit une hypothèse en ajoutant successivement le meilleur prédicat. Les résultats présentés ont aussi été confirmés en utilisant SMART+ [Botta 93] et G-NET [Anglano 98].

Dans la zone YES, tout se passe bien, le concept qui a servi à générer les données est retrouvé rapidement par l'algorithme. Mais au niveau de la transition de phase et dans la zone NO proche de la transition de phase, les algorithmes fournissent au bout d'un temps très long une hypothèse de mauvaise qualité : cette hypothèse a un taux d'erreur voisin de 50% sur les exemples de test. Si on regarde l'hypothèse de plus près, on remarque qu'elle est composée de nombreuses clauses très spécifiques. L'algorithme a donc sur-appris : chaque clause composant l'hypothèse apprise est en réalité une description très précise de quelques exemples.

Enfin, dans la région NO, les algorithmes fournissent une hypothèse qui classe correctement les exemples de test. Mais le temps nécessaire à l'apprentissage de cette hypothèse est plus long que dans la région YES. Et, surtout, même si cette hypothèse partage des prédicats avec le concept à apprendre, elle est différente de ce concept. Si l'objectif est uniquement de fournir un modèle capable de classer de futurs exemples, cela n'est pas problématique. Mais si le but est de « comprendre » le

modèle qui a généré les données, ce but n'est pas atteint.

Des algorithmes adaptés Ces découvertes ont conduit à la conceptions d'algorithmes d'apprentissage adaptés, qui peuvent résoudre certains des problèmes rencontrés.

cBIL [Bianchetti 02] se confronte à la région NO. Cet algorithme suit une stratégie *bottom-up* en favorisant l'apprentissage d'hypothèses complexes. Ainsi, les hypothèses testées se trouvent dans la région NO, i.e. là où se trouve l'hypothèse recherchée. Dans ce cadre, même s'il y a peu de chance de trouver une hypothèse qui couvre correctement tous les exemples, une telle hypothèse peut immédiatement être considérée comme la bonne.

[Alphonse 08b] s'intéresse plus spécifiquement à l'apport d'exemples particuliers : les exemples *near-miss*. Lorsque l'ensemble d'entraînement contient de tels exemples, les plateaux ne posent plus problème à condition d'utiliser une approche *top-down* restreignant les hypothèses testées en fonction des données (on parle d'approche *data-driven*). La difficulté est alors d'utiliser des bases de données incluant des exemples *near-miss*.

3.3 Discussion

Les deux sections précédentes présentent chacune l'étude d'un modèle dans lequel est mis en évidence une transition de phase. Ces deux études suivent le même schéma.

Identification de la zone de transition de phase Tout d'abord des données sont générées selon des paramètres d'ordre. Ces paramètres contrôlent la difficulté de résolution des problèmes générés. Dans l'espace des paramètres d'ordre, plusieurs régions sont alors mises en évidence : une région où la plupart des problèmes sont satisfiables, une région où la plupart des problèmes sont non-satisfiables, et une région de transition entre les deux premières. La propriété intéressante est que le passage de la région satisfiable à la région non-satisfiable est très rapide, ou, en d'autre termes, la région de transition est très fine. Ce comportement, rappellant les changements de phase en physique statistique, est appelé *transition de phase*.

Une zone de transition de phase "difficile" Une deuxième propriété est le fait que la zone de transition est une zone « difficile ». La difficulté est tout d'abord computationelle : il faut plus de temps pour déterminer la satisfiabilité d'un problème présent dans la transition de phase. Mais dans le cas particulier de l'ILP, cette zone s'avère aussi difficile pour l'apprentissage, i.e. il est difficile (voire impossible) d'apprendre une hypothèse appartenant à cette zone.

L'identification de cette zone difficile a un intérêt direct. Une fois la zone difficile connue, la comparaison entre deux algorithmes se fait dans cette zone, i.e. là où le comportement de ces algorithmes est le plus intéressant. Mais, pour aller plus loin, le fait de découper l'espace en plusieurs zones permet de dresser des cartes de compétences des algorithmes. Ces cartes répertorient les zones pour lesquelles un algorithme est compétent, ou permettent d'identifier les algorithmes compétents pour une zone donnée. Ces cartes permettent donc d'espérer répondre à la question du méta-apprentissage [Vilalta 02], concernant le choix de l'algorithme le plus adapté à des données fixées.

Des algorithmes dédiés Enfin, l'étude du comportement des algorithmes dans les diverses régions, permet la conception d'algorithmes adaptés à une région particulière.

Le chapitre suivant se fonde sur ce schéma d'étude pour déterminer les limites du noyau-somme appliqué aux problèmes à instances multiples.

Chapitre 4

Mise en évidence de la Transition de Phase pour les noyaux relationnels

En s'inspirant de la méthodologie définie pour l'étude de la transition de phase, ce chapitre étudie les limites des noyaux-somme présentés en section 2.3.2. Nous définissons tout d'abord une famille de concepts conjonctifs à instances multiples et nous déterminons pour quels sous-ensembles de ces concepts le noyau-somme n'est pas en mesure d'apporter une information. Enfin nous proposons un nouveau critère lié à l'erreur en généralisation permettant de caractériser les concepts pour lesquels les noyaux-somme sont adaptés ou non.

4.1 Position du problème

Les machines à vecteurs de support et l'astuce des noyaux ont permis de grandes avancées en apprentissage propositionnel (section 2.2). Une question encore ouverte concerne les limitations de cette approche dans le cadre relationel (section 2.3), i.e. s'il est possible de définir un noyau entre deux exemples qui permette en toute généralité de reconstruire des informations de type existentiel, universel, ou encore une combinaison des deux.

L'intérêt des problèmes à instances multiples est qu'ils fournissent un intermédiaire entre les problèmes propositionnels et les problèmes relationnels (section 2.1.4). Ces problèmes offrent donc un bon terrain d'étude pour la caractérisation et l'étude des algorithmes relationnels.

Le noyau-somme est justement un noyau développé pour traiter les problèmes à instances multiples (section 2.3.2). Ce noyau est capable notamment de reconstruire des informations de type existentiel mais simples. Formellement, le noyau-somme est en mesure de retrouver un concept classant comme positifs les exemples dont une instance au moins vérifie un prédicat C :

$$\text{positif}(x) \Leftrightarrow \exists I \in x, C(I)$$

L'objectif de ce chapitre est de montrer que le noyau-somme n'est pas toujours en mesure de reconstruire des informations plus complexes. Plus précisément, ce chapitre étudies, sous l'angle théorique, sous quelles conditions le noyau-somme peut retrouver des concepts faisant intervenir plusieurs prédicats $C_1, \ldots, C_P$:

$$\text{positif}(x) \Leftrightarrow \exists I_1 \ldots I_P \in x, \bigwedge_{i=1}^{P} C_i(I_i)$$

L'ilustration expérimentale des comportements et des propriétés identifiés sera donnée au chapitre 5.

4.2 Modèle de génération de données à instances multiples

Comme indiqué précédemment (section 3.1), le canevas de la transition de phase étudie le comportement moyen d'objets générés aléatoirement. Ces objets suivent une loi définie à partir de différents paramètres d'ordre, capturant la difficulté de la tâche.

En appliquant ce schéma aux problèmes à instances multiples, trois types de paramètres d'ordre, liés respectivement aux instances, aux exemples et au concept-cible, sont définis :

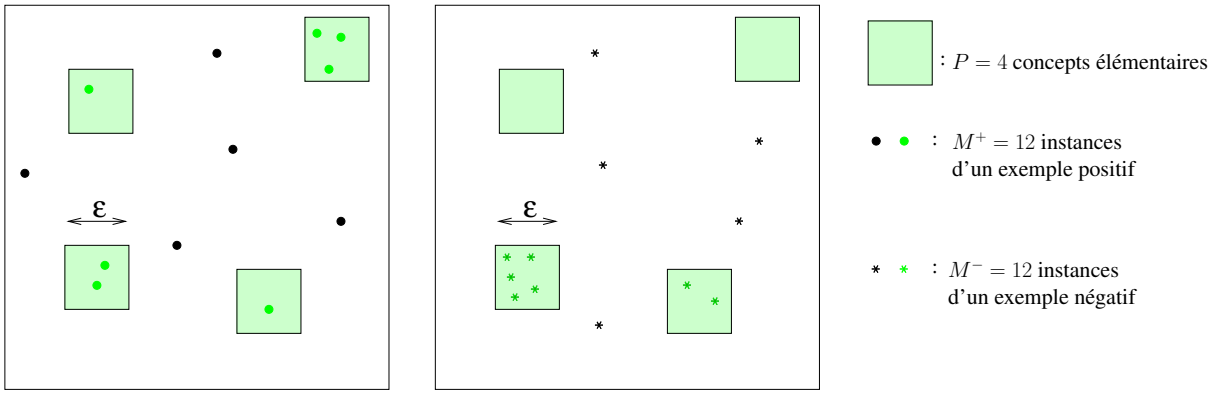
- chaque *instance* $I = (a, \boldsymbol{z})$ est formée d'un symbole a tiré dans un alphabet Σ, et d'un vecteur de dimension d, dans $[0,1]^d$. Par définition, l'ε-boule d'une instance I notée $\mathcal{B}_\varepsilon(I)$ correspond à l'ensemble des instances $I' = (a', \boldsymbol{z}')$ telles que I et I' comportent le même symbole $a = a'$ et telles que la norme infinie $\|\boldsymbol{z} - \boldsymbol{z}'\|_\infty$ entre $\boldsymbol{z}$ et $\boldsymbol{z}'$ soit plus petite que ε (i.e. pour chaque coordonnée k de $\boldsymbol{z}$ et $\boldsymbol{z}'$, $|z_k - z'_k| \leq \varepsilon$) ;
- un *exemple* x_i positif (respectivement négatif) est caractérisé par un ensemble de M^+ (resp. M^-) instances notées $x_{i,1}, \ldots, x_{i,M^+}$ (resp. $x_{i,1}, \ldots, x_{i,M^-}$) ;
- le *concept-cible* est défini par la conjonction de P concepts élémentaires $C_1, \ldots, C_P$, avec C_i l'ε-boule centrée sur une instance I_i. Un exemple x est positif si et seulement si chaque concept élémentaire C_i du concept-cible contient une instance de x.

Les instances du concept-cible sont tirées uniformément dans $[0,1]^d$. Les M^+ instances des *exemples positifs* sont tirées comme suit : m^+ instances sont tirées dans les concepts élémentaires C_i en s'assurant qu'au moins une instance est tirée dans chaque C_i ($m^+ \geq P$) ; les $M^+ - m^+$ autre instances sont tirées de façon uniforme dans $[0,1]^d$. De la même manière, les M^- instances des exemples négatifs comprennent m^- instances tirées dans les concepts élémentaires C_i, en s'assurant que P_m (m pour *marge*) C_i ne sont pas visités ($P_m \geq 1$) ; les autres $M^- - m^-$ instances sont tirées uniformément dans $[0,1]^d$. Les instances appartenant à un concept élémentaire sont dites *pertinentes*, puisque c'est à partir de ces instances uniquement qu'est définie la classe d'un exemples.

Le tableau 4.1 résume ses paramètres d'ordres et la figure 4.1 représente un exemple positif et un exemple négatif.

Concept univers En l'absence de contraintes, les instances sont tirées uniformément dans $[0,1]^d$. Or les domaines d'application font souvent intervenir certaines contraintes sur la distribution des instances, s'opposant ainsi à une distribution uniforme. Pour modéliser ce type de comportement, nous introduisons la notion de *concept univers*. L'univers est un ensemble de Q ε-boules ; les instances sont alors tirées soit dans un concept élémentaire, soit dans une boule de l'univers.

Par symétrie avec le concept-cible, nous imposons à chaque exemple positif de ne pas visiter Q_m boules de l'univers.



(a) Instances d'un exemple positif (b) Instances d'un exemple négatif

FIG. 4.1 – Un exemple positif (a) et un exemple négatif (b) générés selon les paramètres d'ordres suivant : $|\Sigma| = 0$, $d = 2$, $M^+ = M^- = 12$, $m^+ = m^- = 7$, $P_m = 2$, $P = 4$. Chacun des quatres concepts élémentaires est représenté par un carré. Pour être positif, un exemple doit avoir au moins une instance présente dans chaque carré.

catégorie	paramètre	définition
instances $I = (a, \boldsymbol{z})$	$\|\Sigma\|$	taille de l'alphabet auquel appartient l'attribut discret d'une instance, $a \in \Sigma$
	d	dimension des attributs réels d'une instance, $\boldsymbol{z} \in \mathbb{R}^d$
exemples	M^+	nombre d'instances composant un exemple positif
	M^-	nombre d'instances composant un exemple négatif
	m^+	nombre d'instances appartenant à un concept élémentaire pour un exemple positif
	m^-	nombre d'instances appartenant à un concept élémentaire pour un exemple négatif
	P_m	nombre de concepts élémentaires « manqués » par un exemple négatif
	Q_m	nombre de boules de l'univers « manquées » par un exemple positif
concept	P	nombre de concepts élémentaires composant le concept-cible
	ε	rayon des concepts élémentaires
univers	Q	nombre de ε-boules composant l'univers

TAB. 4.1 – Résumé des paramètres d'ordre pour un problème d'apprentissage à instances multiples.

4.3 Limites théoriques du noyau-somme

Le noyau-somme permet d'identifier un concept existentiel simple (section 2.3.2) : un exemple est positif si l'une de ses instances vérifie un concept élémentaire. Cette section s'intéresse aux propriétés du noyau-somme dans le cas où le concept-cible est conjonctif : un exemple est positif si ses instances lui permettent de vérifier plusieurs concepts élémentaires. L'étude théorique proposée, restreinte aux données générées selon le schéma fourni dans la section 4.2, a pour but de caractériser dans quelles conditions un noyau-somme est / n'est pas en mesure de discriminer les exemples positifs des exemples négatifs.

4.3.1 Positionnement

Soit k un noyau sur les instances et K le noyau-somme correspondant (équation 2.12). L'utilisation de K revient à propositionnaliser la représentation des exemples en associant à chaque exemple x le vecteur de $\mathbb{R}^n$:

$$\Phi(x) = (K(x_1, x), \ldots, K(x_n, x))$$

où $x_1, \ldots, x_n$ sont les exemples d'entraînement. Plus précisément, une machine à vecteurs de support utilisant K comme noyau apprendra un concept fondé sur une séparation linéaire de $\mathbb{R}^n$:

$$\text{classe}(x) = \text{signe}(h(x))$$

avec

$$\begin{aligned} h(x) &= \sum_{i=1}^{n} \alpha_i K(x_i, x) + b \\ &= \langle \alpha, \Phi(x) \rangle + b \end{aligned}$$

La figure 4.2 montre la distribution de la représentation $\Phi(x)$ associée à un ensemble d'entraînement composé d'un exemple positif et d'un exemple négatif. On remarque que les exemples négatifs et les exemples positifs forment deux nuages de points. Avec les paramètres choisis, ces deux nuages sont distincts et séparables par une droite, il existe donc une hypothèse de type machine à vecteurs de support classant parfaitement tous ces exemples.

Plus généralement, pour pouvoir séparer linéairement les exemples positifs des exemples négatifs dans l'espace propositionnel, il faut que la valeur moyenne prise par certains attributs diffère suivant que l'on considère les exemples positifs ou les exemples négatifs. Nous allons donc déterminer ces valeurs moyennes pour identifier les paramètres d'ordre qui rendent le noyau-somme non-informatif.

4.3.2 Modèle génératif simple

Nous commençons par étudier le comportement de K dans le cas de modèles sans univers, i.e. lorsque les attributs réels des instances non-pertinentes sont tirés uniformément dans $[0, 1]$ (section 4.2).

Valeur moyenne prise par le noyau-somme Soit x_i l'un des exemples d'entraînement. L'apport principal de cette section est de déterminer l'espérance de $K(x_i, x)$ lorsque l'on fait varier x (théorème 1). L'expression de cette espérance diffère suivant la classe de x_i et de x.

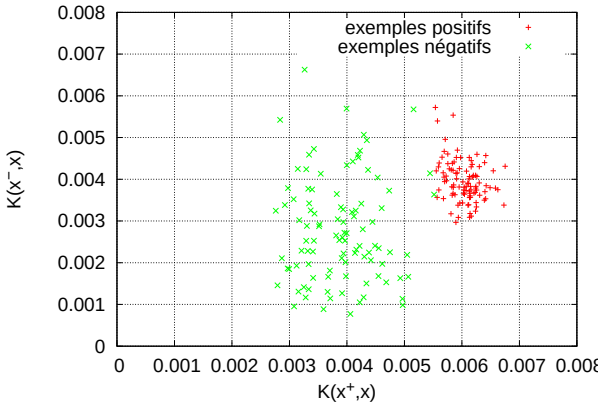


FIG. 4.2 – Distribution de la représentation $\Phi(x)$ obtenue à partir d'un ensemble d'entraînement composé d'un exemple positif x^+ et d'un exemple négatif x^-. Chaque point correspond à la représentation associée à l'un des cent exemples positifs (représentés par $+$) ou l'un des cent exemples négatifs (représentés par $\times$). Les paramètres d'ordre régissant la génération des exemples sont fixés à $|\Sigma| = 15$, $d = 30$, $M^+ = M^- = 100$, $m^+ = 50$, $m^- = 30$, $P_m = 20$, $P = 30$, $\varepsilon = 0.15$.

THÉORÈME **1.** *Soit D^+ (respectivement D^-) la loi de distribution des exemples positifs (resp. négatifs), et x_i un exemple d'entraînement. Soit ν_0 (respectivement ν_1) la loi de $k(I, I')$ lorsque I et I' sont deux instances quelconques (resp. deux instances appartenant au même concept élémentaire). Notons μ_0 et μ_1 les espérances respectives de ν_0 et ν_1. Alors*

$$\mathbb{E}_{x\sim D^+}\left[K(x_i,x)\right] = \begin{cases} \left(\frac{m^+}{M^+}\right)^2 \frac{\mu_1-\mu_0}{P} + \mu_0 & \text{si } x_i \text{ est positif} \\ \frac{m^+}{M^+}\frac{m^-}{M^-}\frac{\mu_1-\mu_0}{P} + \mu_0 & \text{si } x_i \text{ est négatif} \end{cases} \tag{4.1a}$$

$$\mathbb{E}_{x\sim D^-}\left[K(x_i,x)\right] = \begin{cases} \frac{m^+}{M^+}\frac{m^-}{M^-}\frac{\mu_1-\mu_0}{P} + \mu_0 & \text{si } x_i \text{ est positif} \\ \left(\frac{m^-}{M^-}\right)^2 \frac{\mu_1-\mu_0}{P} + \mu_0 & \text{si } x_i \text{ est négatif} \end{cases} \tag{4.1b}$$

Démonstration. Les démonstrations des quatre égalités du théorème 1 sont similaires, nous nous intéresserons donc uniquement au calcul de $\mathbb{E}_{x_j\sim D^+}\left[K(x_i,x_j)\right]$.

On note C_{P+1} l'ensemble des instances qui n'appartiennent pas à un concept élémentaire et $\boldsymbol{m}(x)$ le vecteur de $\mathbb{N}^{P+1}$ dont la k-ième coordonnée $m_k(x)$ est le nombre d'instances de x appartenant à C_j. De par la manière dont sont générées les données, on a alors pour tout exemple positif x

$$\sum_{j=1}^{P} m_j(x) = m^+ \qquad m_{P+1}(x) = M^+ - m^+ \qquad \sum_{j=1}^{P+1} m_j(x) = M^+$$

et pour tout exemple négatif x'

$$\sum_{j=1}^{P} m_j(x') = m^- \qquad m_{P+1}(x') = M^- - m^- \qquad \sum_{j=1}^{P+1} m_j(x') = M^-$$

De plus, comme les m^+ et m^- instances qui appartiennent à un concept élémentaire sont tirées uniformément parmi les concepts élémentaires, l'espérance de $m_j(x)$ pour $j = 1, \ldots, P$ est de $\frac{m^+}{P}$ pour un exemple positif et $\frac{m^-}{P}$ pour un exemple négatif.

Avec ces notations et x_i un exemple positif,

$$
\begin{aligned}
&\mathbb{E}_{x_j \sim D^+}\left[K(x_i, x_j)\right] \\
&= \mathbb{E}_{x_j \sim D^+}\left[\frac{1}{M^+}\frac{1}{M^+}\sum_{k=1}^{M^+}\sum_{\ell=1}^{M^+} k(x_{i,k}, x_{j,\ell})\right] \\
&= \left(\frac{1}{M^+}\right)^2 \mathbb{E}_{x_j \sim D^+}\left[\sum_{r=1}^{P}\sum_{x_{i,k}\in C_r}\sum_{x_{j,\ell}\in C_r} k(x_{i,k}, x_{j,\ell}) + \sum_{\substack{(s,r)\in[|1,P+1|]^2 \\ s\neq r \vee s=r=P+1}}\sum_{x_{i,k}\in C_r}\sum_{x_{j,\ell}\in C_s} k(x_{i,k}, x_{j,l})\right] \\
&= \left(\frac{1}{M^+}\right)^2 \mathbb{E}_{\boldsymbol{m}(x_j)\sim\boldsymbol{m}(D^+)}\left[\sum_{r=1}^{P} m_r(x_i)m_r(x_j)\mu_1 + \sum_{\substack{(s,r)\in[|1,P+1|]^2 \\ s\neq r \vee s=r=P+1}} m_r(x_i)m_s(x_j)\mu_0\right] \\
&= \left(\frac{1}{M^+}\right)^2 \mathbb{E}_{\boldsymbol{m}(x_j)\sim\boldsymbol{m}(D^+)}\left[\sum_{r=1}^{P} m_r(x_i)m_r(x_j)(\mu_1-\mu_0) + \sum_{(s,r)\in[|1,P+1|]^2} m_r(x_i)m_s(x_j)\mu_0\right] \\
&= \left(\frac{1}{M^+}\right)^2 \mathbb{E}_{\boldsymbol{m}(x_j)\sim\boldsymbol{m}(D^+)}\left[\sum_{r=1}^{P} m_r(x_i)m_r(x_j)(\mu_1-\mu_0) + \sum_{r=1}^{P+1} m_r(x_i)\sum_{s=1}^{P+1} m_s(x_j)\mu_0\right] \\
&= \left(\frac{1}{M^+}\right)^2 \mathbb{E}_{\boldsymbol{m}(x_j)\sim\boldsymbol{m}(D^+)}\left[\sum_{r=1}^{P} m_r(x_i)m_r(x_j)(\mu_1-\mu_0) + M^+M^+\mu_0\right] \\
&= \left(\frac{1}{M^+}\right)^2 \left(\sum_{r=1}^{P} m_r(x_i)\,\mathbb{E}_{\boldsymbol{m}(x_j)\sim\boldsymbol{m}(D^+)}\left[m_r(x_j)\right](\mu_1-\mu_0) + M^+M^+\mu_0\right) \\
&= \left(\frac{1}{M^+}\right)^2 \left(\sum_{r=1}^{P} m_r(x_i)\frac{m^+}{P}(\mu_1-\mu_0) + M^+M^+\mu_0\right) \\
&= \left(\frac{1}{M^+}\right)^2 \left(m^+\frac{m^+}{P}(\mu_1-\mu_0) + M^+M^+\mu_0\right) \\
&= \left(\frac{m^+}{M^+}\right)^2 \frac{\mu_1-\mu_0}{P} + \mu_0
\end{aligned}
$$

□

Zone d'échec du noyau-somme Le théorème 1 permet de déterminer la valeur qui nous intéresse : la différence entre l'espérance de $K(x_i, x)$ pour x positif et la même espérance pour x négatif.

COROLLAIRE **1.** *Soit D^+ (respectivement D^-) la loi de distribution des exemples positifs (resp. négatifs), et x_i un exemple d'entraînement. Soit ν_0 (respectivement ν_1) la loi de $k(I, I')$ lorsque I et I' sont deux instances quelconques (resp. deux instances appartenant au même concept élémentaire).*

Notons μ_0 et μ_1 les espérances respectives de ν_0 et ν_1. Alors

$$\begin{aligned}\Delta_K(x_i) &\stackrel{\text{def}}{=} \mathbb{E}_{x\sim D^+}\left[K(x_i,x)\right] - \mathbb{E}_{x\sim D^-}\left[K(x_i,x)\right] \\ &= \begin{cases} \left(\frac{m^+}{M^+} - \frac{m^-}{M^-}\right)\frac{m^+}{M^+}\frac{\mu_1-\mu_0}{P} & \textit{si } x_i \textit{ est positif} \\ \left(\frac{m^+}{M^+} - \frac{m^-}{M^-}\right)\frac{m^-}{M^-}\frac{\mu_1-\mu_0}{P} & \textit{si } x_i \textit{ est négatif} \end{cases}\end{aligned} \tag{4.2}$$

Le noyau-somme ne permet donc pas de différencier les exemples positifs des exemples négatifs dans un contexte linéaire si $\frac{m^+}{M^+}$ et $\frac{m^-}{M^-}$ sont égaux. Ainsi, le noyau-somme se base principalement sur les proportions d'instances pertinentes pour différencier les exemples positifs des exemples négatifs. Cette différence de proportion est pourtant éloignée du concept réel : une différence de distribution.

De plus, si $\Delta_K(x_i)$ est positif mais petit, l'information apportée par le noyau-somme sera trop faible et l'algorithme d'apprentissage ne pourra pas trouver un bon séparateur linéaire. C'est justement sur cette différence que jouent les paramètres μ_0 et μ_1 qui dépendent à la fois du noyau k utilisé sur les instances, du rayon ε des concepts élémentaires, de la taille de l'alphabet Σ, et de la dimension d de l'espace des instances. Ainsi, plus l'écart entre μ_0 et μ_1 est faible, plus la séparation entre exemples négatifs et exemples positifs est difficile. De la même manière, $\Delta_K(x_i)$ diminue inversement proportionnellement au nombre P de concepts élémentaires composant le concept cible. Donc un concept comportant de nombreux concepts élémentaires sera difficile à estimer à l'aide du noyau-somme.

Enfin, l'influence de P_m ne se voit pas directement avec l'expression de $\Delta_K(x_i)$. À première vue, plus le nombre P_m de concepts cibles « manqués » par un exemple négatif est élevé, plus il est facile de faire la différence entre un exemple négatif et un exemple positif. C'est pourtant le contraire qui se produit : en augmentant P_m, on augmente la variance de $K(x_i,x)$ tout en laissant $\Delta_K(x_i)$ constant, et par suite on rend plus difficile la séparation entre exemples positifs et exemples négatifs. Ce résultat contre-intuitif confirme le problème principal du noyau-somme utilisé dans un contexte linéaire : au lieu de détecter des propriétés existentielles, ce noyau recherche une différence du coté de la proportion d'instances pertinentes.

4.3.3 Modèle génératif avec univers

Cette section s'intéresse à l'effet de l'univers sur les résultats présentés dans la section 4.3.2. Tout d'abord, le théorème 1 est modifié et fait intervenir à présent la taille de l'univers. Ce changement est dû au fait que deux instances tirées dans la même boule de l'univers n'auront en moyenne pas la même différence (calculée par le noyau k sur les instances) que deux instances quelconques. Le théorème devient alors :

THÉORÈME **2.** *Soit D^+ (respectivement D^-) la loi de distribution des exemples positifs (resp. négatifs), et x_i un exemple d'entraînement. Soit ν_0 (respectivement ν_1) la loi de $k(I,I')$ lorsque I et I' sont deux instances quelconques (resp. deux instances appartenant au même concept élémentaire). Notons μ_0 et μ_1 les espérances respectives de ν_0 et ν_1. Alors*

$$\begin{aligned}&\mathbb{E}_{x\sim D^+}\left[K(x_i,x)\right] \\ &\quad= \begin{cases} \left(\left(\frac{m^+}{M^+}\right)^2\frac{1}{P} + \left(1-\frac{m^+}{M^+}\right)^2\frac{1}{Q}\right)(\mu_1-\mu_0)+\mu_0 & \textit{si } x_i \textit{ est positif} \\ \left(\frac{m^+}{M^+}\frac{m^-}{M^-}\frac{1}{P} + \left(1-\frac{m^+}{M^+}\right)\left(1-\frac{m^-}{M^-}\right)\frac{1}{Q}\right)(\mu_1-\mu_0)+\mu_0 & \textit{si } x_i \textit{ est négatif} \end{cases}\end{aligned} \tag{4.3a}$$

$$\mathbb{E}_{x\sim D^-}\left[K(x_i,x)\right] = \begin{cases} \left(\frac{m^-}{M^-}\frac{m^+}{M^+}\frac{1}{P} + \left(1-\frac{m^-}{M^-}\right)\left(1-\frac{m^+}{M^+}\right)\frac{1}{Q}\right)(\mu_1-\mu_0)+\mu_0 & \textit{si } x_i \textit{ est positif} \\ \left(\left(\frac{m^-}{M^-}\right)^2\frac{1}{P} + \left(1-\frac{m^-}{M^-}\right)^2\frac{1}{Q}\right)(\mu_1-\mu_0)+\mu_0 & \textit{si } x_i \textit{ est négatif} \end{cases} \tag{4.3b}$$

Remarques Tout d'abord, on remarque que l'effet de l'univers dépend de la proportion d'instances pertinentes pour un exemple. Lorsque les exemples sont principalement composés d'instances pertinentes, c'est la taille P du concept cible qui influence le plus l'espérance de K. Au contraire, lorsque les exemples contiennent peu d'instances pertinentes, l'espérance de K dépend principalement de la taille Q de l'univers.

On peut aussi remarquer que l'on retrouve bien le théorème 1 lorsque l'on fait tendre Q vers l'infini dans le théorème 2.

Zone d'échec du noyau-somme L'espérance de K en présence d'un univers modifie de manière correspondante la zone d'échec du noyau-somme.

Corollaire **2.** *Soit D^+ (respectivement D^-) la loi de distribution des exemples positifs (resp. négatifs), et x_i un exemple d'entraînement. Soit ν_0 (respectivement ν_1) la loi de $k(I,I')$ lorsque I et I' sont deux instances quelconques (resp. deux instances appartenant au même concept élémentaire). Notons μ_0 et μ_1 les espérances respectives de ν_0 et ν_1. Alors*

$$\begin{aligned}\Delta_K(x_i) &\stackrel{\text{def}}{=} \mathbb{E}_{x\sim D^+}\left[K(x_i,x)\right] - \mathbb{E}_{x\sim D^-}\left[K(x_i,x)\right] \\ &= \begin{cases} \left(\frac{m^+}{M^+}-\frac{m^-}{M^-}\right)(\mu_1-\mu_0)\left(\frac{m^+}{M^+}\frac{1}{P} - (1-\frac{m^+}{M^+})\frac{1}{Q}\right) & \textit{si } x_i \textit{ est positif} \\ \left(\frac{m^+}{M^+}-\frac{m^-}{M^-}\right)(\mu_1-\mu_0)\left(\frac{m^-}{M^-}\frac{1}{P} - (1-\frac{m^-}{M^-})\frac{1}{Q}\right) & \textit{si } x_i \textit{ est négatif} \end{cases}\end{aligned} \tag{4.4}$$

Comme pour le cas sans univers, le noyau-somme ne permet donc pas de différencier les exemples positifs des exemples négatifs dans un contexte linéaire si $\frac{m^+}{M^+}$ et $\frac{m^-}{M^-}$ sont égaux. Mais en dehors de ces valeurs, l'effet de l'univers par rapport au cas sans univers dépend de $\frac{m^+}{M^+}$ et $\frac{m^-}{M^-}$.

Lorsque les exemples sont majoritairement composés d'instances pertinentes ($\frac{m^+}{M^+} \approx 1$ et $\frac{m^-}{M^-} \approx 1$), les termes $(1-\frac{m^+}{M^+})\frac{1}{Q}$ et $(1-\frac{m^-}{M^-})\frac{1}{Q}$ deviennent négligeables et $\Delta_K(x_i)$ retrouve les valeurs données par le corrolaire 1. En d'autres termes, tout se passe comme s'il n'y avait pas d'univers.

Au contraire, lorsque les exemples sont principalement composés d'instances non-pertinentes, c'est le terme provenant de l'univers qui est déterminant. Le noyau-somme permet alors de différencier les exemples à partir des instances non-pertinentes, bien que ces instances soient censées ne fournir aucune information.

La figure 4.3 montre l'évolution du ratio entre la valeur absolue de $|\Delta_K|$ avec univers et la valeur absolue de $|\Delta_K|$ sans univers en fonction de la proportion r d'instances pertinentes pour un exemple donné. Cette courbe permet de visualiser l'effet de l'univers, i.e. de comprendre quand l'univers favorise ou non la séparation entre exemples positifs et exemples négatifs.

Ainsi, pour r compris entre 0 et $\frac{P}{P+2Q}$, l'univers accroît la différence entre exemples négatifs et exemples positifs, aidant ainsi à les différencier. Au contraire, pour r compris entre $\frac{P}{P+2Q}$ et 1, l'introduction de l'univers rend plus difficile la séparation entre exemples négatifs et exemples positifs. Toutefois, lorsque r est suffisamment proche de 1, l'effet de l'univers est négligeable. Enfin, pour r proche de $\frac{P}{P+Q}$, Δ_K est proche de zéro, ainsi il devient très difficile de distinguer les exemples positifs des exemples négatifs.

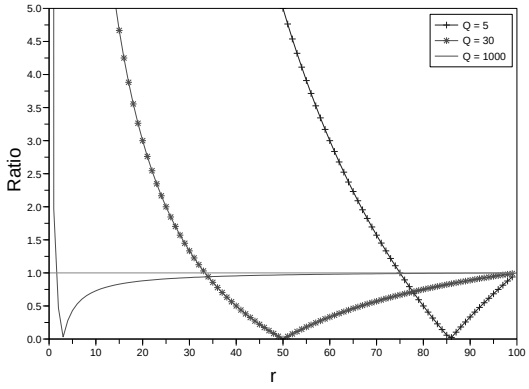


FIG. 4.3 – Ratio entre $|\Delta_K(x_i)|$ avec univers et $|\Delta_K(x_i)|$ sans univers en fonction de la proportion r d'instances pertinentes pour x_i. Ces courbes ne dépendent pas de la classe de x_i ni de la proportion d'instances pertinentes pour les exemples de la classe opposée à celle de x_i. Le nombre P de concepts cible est fixé à 30 et le nombre Q de boules définissant l'univers varie dans $\{5, 30, 1\ 000\}$

4.3.4 Remarque

$\Delta_K(x_i)$ informe uniquement sur la possibilité de séparer linéairement les exemples négatifs des exemples positifs dans la représentation induite par Φ. Mais ce n'est pas pour autant que K ne fournit aucune information sur l'étiquette des exemples. En effet, les exemples négatifs augmentent la variance de $K(x_i, x)$ et les valeurs extrêmes de $K(x_i, x)$, pour x_i fixé, sont atteintes par des exemples négatifs. Donc l'hypothèse h définie par :

$$h(x) = \exists x_i \in \mathcal{E} \text{ t.q. } K(x_i, x) \text{ est une valeur extrême}$$

sera en mesure de classer les exemples à condition que $\mathcal{E}$ soit suffisamment varié.

4.4 Un nouveau critère d'apprentissage

Pour vérifier concrètement l'influence des paramètres d'ordre sur l'information apportée par le noyau-somme pour la classification, une approche standard consiste à regarder l'erreur en test obtenue **par un algorithme ou un ensemble d'algorithmes donné** sur des problèmes artificiels générés selon ces paramètres d'ordre. Chaque problème est composé d'un ensemble d'entraînement à partir duquel est apprise une hypothèse $\hat{h}$ et d'un ensemble de test sur lequel est calculée l'erreur de $\hat{h}$. L'erreur en test, moyennée sur plusieurs problèmes générés avec les mêmes paramètres d'ordre, mesure en effet la compétence d'un algorithme conditionnellement à la valeur des paramètres [Botta 03].

Cette thèse suit une approche différente, pour les raisons suivantes. Notre but est d'examiner à quel point l'astuce du noyau peut être utilisée pour résoudre les difficultés spécifiques à l'apprentissage relationnel ; en termes relationnels, la question porte sur la qualité de la représentation obtenue au travers du noyau relationnel. En d'autres termes, l'accent est mis sur la représentation (le potentiel de l'espace de recherche des hypothèses définies par le noyau pour instances multiples) et

non sur un algorithme particulier (la qualité de la meilleure hypothèse proposée par cet algorithme dans cet espace de recherche).

En conséquence, la méthodologie suivie est fondée sur la génération de problèmes artificiels composés d'un ensemble d'entraînement $\mathcal{E} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ et d'un ensemble de test $\mathcal{T} = (x'_1, y'_1), \ldots, (x'_{n'}, y'_{n'})$. L'ensemble d'entraînement induit une propositionnalisation du domaine, en associant à chaque exemple à instances multiples x le vecteur réel de dimension n :

$$\Phi(x) = (K(x_1, x), \ldots, K(x_n, x))$$

Notons $\mathcal{R}_\mathcal{E}$ cette représentation propositionnelle fondée sur l'ensemble d'entraînement $\mathcal{E}$.

La nouveauté de la méthodologie proposée est le traitement du problème d'apprentissage à instances multiples comme un problème de satisfaction de contrainte dans l'espace de représentation $\mathcal{R}_\mathcal{E}$. La question est de savoir s'*il existe une hypothèse* définie sur $\mathcal{R}_\mathcal{E}$ capable de classer correctement les exemples de l'ensemble de test, et non pas s'*il existe un algorithme* capable de trouver une telle hypthèse.

Formellement, nous examinons l'existence d'une forme linéaire h définie sur $\mathbb{R}^n$ telle que h appartienne à l'espace de recherche exploré par une machine à vecteurs de supports utilisant le noyau-somme. h s'écrit donc sous la forme

$$h(z) = \sum_{i=1}^{n} \alpha_i y_i z_i + b = \langle \beta, z \rangle + b$$

avec, pour tout i, α_i positif et $\beta_i = \alpha_i y_i$. De plus nous imposons à h de séparer les exemples de test projetés dans $\mathcal{R}_\mathcal{E}$; pour tout exemple de test x'_j d'étiquette y'_j, $h(\Phi(x'_j))$ est du signe de y'_j.

En d'autres termes, la question examinée est (Q1) : *existe-t-il un hyper-plan séparateur dans la représentation propositionnelle $\mathcal{R}_\mathcal{E}$ définie à partir de l'ensemble d'entraînement, qui appartienne à l'espace de recherche de la machine à vecteurs de support utilisant le noyau-somme (équations Q1b et Q1c), et qui classe correctement les exemples de test (équation Q1a) ?* Cette question doit être opposée à (Q2) : *l'hyper-plan séparateur, qui aurait été appris à partir des données d'entraînement par la machine à vecteurs de support utilisant le noyau-somme, classe-t-il correctement les exemples de test ?*

$$\begin{array}{l} \text{Trouver } \alpha \in \mathbb{R}^n, b \in \mathbb{R} \\ \text{t.q. } \begin{cases} y'_j(\langle \beta, \Phi(x'_j) \rangle + b) \geq 1, & j = 1 \ldots n' \quad \text{(Q1a)} \\ \alpha_i \geq 0, & i = 1 \ldots n \quad \text{(Q1b)} \\ \beta_i = \alpha_i y_i, & i = 1 \ldots n \quad \text{(Q1c)} \end{cases} \end{array} \tag{Q1}$$

Clairement, la question (Q1) est moins contrainte que (Q2) puisque (Q1) « triche » en utilisant les exemples de test pour déterminer les coefficient α_i et b. Le problème de satisfaction de contrainte (Q1) fournit une information plus précise sur la qualité de la propositionnalisation fondée sur l'astuce des noyaux. Formellement, en s'inspirant de [Kearns 93], nous montrons que la probabilité pour (Q1) d'être satisfiable induit une borne inférieure sur l'erreur en généralisation atteignable avec la représentation $\mathcal{R}_\mathcal{E}$.

THÉORÈME **3.** *Soit $p^\star$ l'erreur en généralisation du classifieur linéaire optimal $h^\star$ défini sur $\mathcal{R}_\mathcal{E}$, et soit $\hat{\tau}$ la fraction de problèmes de satisfaction de contraintes* (Q1) *définis à partir de $\mathcal{E}$ qui sont satisfiables parmi T ensembles de test indépendants générés aléatoirement.*

Soit $\eta > 0$, avec probabilité au moins $1 - \exp(-2\eta^2 T)$,

$$p^\star > 1 - (\hat{\tau} + \eta)^{\frac{1}{n'}}$$

Démonstration. La probabilité pour un ensemble $\mathcal{T}$ composé de n' exemples de test de n'inclure aucun exemple mal-classé par $h^\star$ est $(1-p^\star)^{n'}$.

Parallèlement, il est clair que si $\mathcal{T}$ ne contient aucun exemple mal classé par $h^\star$, (Q1) est satisfiable pour $\mathcal{T}$.

Ainsi la probabilité τ que (Q1) soit satisfiable est supérieure à $(1-p^\star)^{n'}$.

$$\tau > (1-p^\star)^{n'}$$

Or, d'après l'inégalité de Hoeffding, la probabilité τ que (Q1) soit satisfiable peut être bornée en fonction de $\hat{\tau}$:

$$\mathbb{P}\left(|\tau - \hat{\tau}| < \eta\right) > 1 - \exp\left(-2\eta^2 T\right)$$

Il en découle qu'avec probabilité $1 - \exp(-2\eta^2 T)$,

$$\begin{aligned} & & \tau &< \hat{\tau} + \eta \\ \text{d'où} & & (1-p^\star)^{n'} &< \hat{\tau} + \eta \\ \text{i.e.} & & p^\star &> 1 - (\hat{\tau} + \eta)^{\frac{1}{n'}} \end{aligned}$$

□

4.5 Résumé

Nous avons proposé un ensemble de paramètres d'ordre (tableau 4.1) pour la génération de problèmes à instances multiples dont le concept sous-jacent est

$$\text{positif}(x) \Leftrightarrow \exists I_1 \dots I_P \in x, \bigwedge_{i=1}^{P} C_i(I_i)$$

où x est un exemple à instances multiples et $C_1, \dots, C_P$ sont des prédicats sur les instances.

Une étude théorique montre que le noyau-somme n'est pas en mesure de retrouver ces concepts si la proportion d'instances pertinentes est la même pour les exemples positifs et négatifs :

$$\frac{m^+}{M^+} = \frac{m^-}{M^-}$$

L'étude se fonde sur l'écart $\Delta_K(x_i)$ entre la valeur prise par le noyau pour un exemple positif et pour un exemple négatif. Plus cet écart est élevé, plus la classification est aisée.

L'étude montre aussi que le noyau k sur les instances ainsi que les paramètres d'ordre autres que m^+, M^+, m^-, et M^- influencent le rapport signal sur bruit, i.e. qu'ils modifient soit $\Delta_K(x_i)$, soit la variance de la valeur prise par le noyau. Ainsi, ces paramètres influent sur la taille de la bande autour de $\frac{m^+}{M^+} = \frac{m^-}{M^-}$ pour laquelle le noyau-somme sera mis en échec.

Afin d'observer concrètement les effets des hyper-paramètres, nous introduisons un *critère de puissance de représentation*, mesurant la probabilité pour un problème à instances multiples donné d'être satisfiable par le biais de la représentation $\mathcal{R}_\mathcal{E}$ induite par le noyau-somme. Cette approche

contraste avec l'approche transition de phase usuelle, qui détermine l'espérance d'échec d'un algorithme donné ; par opposition, ce critère de puissance de représentation s'intéresse à la probabilité d'existence d'une solution.

Le critère ainsi défini induit une borne inférieure sur l'erreur en généralisation qui permet de déterminer *a priori*, et de manière optimiste, la région de compétence d'un algorithme se confrontant aux problèmes à instances multiples à l'aide du noyau-somme.

Le chapitre suivant montre comment ce critère varie empiriquement en fonction de divers paramètres.

Chapitre 5

Résultats expérimentaux

Les chapitres précédents ont situé et caractérisé les limites des noyaux-somme relationels dans le cadre des problèmes à instances multiples. Ce chapitre illustre expérimentalement les résultats négatifs obtenus.

Après avoir décrit le déroulement des expériences, cette section rapporte les résultats obtenus lors d'une large campagne de tests. Ces tests consistent en la génération de problèmes artificiels à instances multiples et en l'observation du pourcentage de problèmes satisfiables en fonction des paramètres d'ordre fixés pour la génération de ces problèmes. La section 5.3 montre que les conclusions déduites du pourcentage de problèmes satisfiables se confirment en terme d'erreur en généralisation.

5.1 Protocole expérimental

Les problèmes à instances multiples sont générés selon le canevas présenté en section 4.2. Les expériences considèrent des valeurs fixes pour tous les paramètres d'ordre, excepté les deux paramètres contrôlant le nombre d'instances pertinentes pour chaque exemple (tableau 5.1).

Instances	$\lvert\Sigma\rvert$	15	d	30
Exemples	M^+, M^-	100	P_m	20
	m^+	$[30, 100]$	m^-	$[10, 100]$
Concept	P	30	ε	0.15
Base de données	n	60 (30 +, 30 −)	n'	200 (100 +, 100 −)
Univers	Q	30	Q_m	0

Tab. 5.1 – Paramètres d'ordre pour la génération des données à instances multiples et leurs valeurs.

Paramètres d'ordre Les instances sont tirées dans $\Sigma \times [0,1]^{30}$ avec $|\Sigma| = 15$. Les instances sont comparées entre elles à l'aide du noyau gaussien :

$$k\left((a,\boldsymbol{z}),(a',\boldsymbol{z}')\right) = \begin{cases} 0 & \text{si } a \neq a' \\ \exp\left(-\|\boldsymbol{z}-\boldsymbol{z}'\|_2^2\right) & \text{sinon} \end{cases}$$

Les exemples sont composés de 100 instances, qu'ils soient positifs ou négatifs, et les exemples négatifs manquent $P_m = 20$ concepts élémentaires. Le nombre m^+ (respectivement m^-) d'instances pertinentes pour les exemples positifs (resp. négatifs) varie dans $[30, 100]$ (resp. $[0, 100]$).

Le concept-cible est la conjonction de $P = 30$ concepts élémentaires $\mathcal{B}_\varepsilon(I_i)$, avec $\varepsilon = 0.15$ et I_i est tiré uniformément dans $\Sigma \times [0,1]^{30}$.

Chaque ensemble d'entraînement $\mathcal{E}$ est composé de 30 exemples positifs et 30 exemples négatifs ($n = 60$) ; chaque ensemble de test $\mathcal{T}$ est composé de 100 exemples positifs et 100 exemples négatifs ($n = 200$).

Enfin, lorsque les instances sont tirées dans l'univers, l'univers est défini par 30 boules dont 15 ne sont pas visitées par chaque exemple positif.

Le tableau 5.1 liste les paramètres d'ordre et leurs valeurs.

Performances mesurées Pour chaque jeu de paramètres d'ordre, quarante problèmes à instances multiple indépendants sont générés. Chaque problème est composé d'un ensemble d'entraînement $\mathcal{E}$ et d'un ensemble de test $\mathcal{T}$. Le problème de satisfaction de contraintes (Q1) correspondant, qui fait appel à $n' = 200$ contraintes de test et $n + 1 = 61$ variables, est construit et résolu en utilisant la bibliothèque GLPK.[1] À chaque jeu de paramètres est associée la proportion de problèmes de satisfaction de contraintes ainsi générés qui sont satisfiables.

L'objectif des expériences est d'examiner comment cette satisfiabilité moyenne, et donc l'intérêt de la propositionnalisation induite par le noyau-somme, varie avec la valeur des paramètres d'ordre. La question est (1) de vérifier que, comme le prédit le chapitre 4, la satisfiabilité moyenne de (Q1) est faible lorsque $\frac{m^+}{M^+}$ et $\frac{m^-}{M^-}$ sont égaux, et (2) d'observer l'évolution de cette satisfiabilité moyenne lorsque $\frac{m^+}{M^+}$ et $\frac{m^-}{M^-}$ ont des valeurs proches.

Les expériences s'intéressent donc plus particulièrement à l'évolution de cette satisfiabilité moyenne en fonction de $\frac{m^+}{M^+}$ et $\frac{m^-}{M^-}$. En pratique, on fait varier différents paramètres d'ordre et on observe l'effet de ces variations sur la taille de la zone d'échec, i.e. de la zone autour de $\frac{m^+}{M^+} = \frac{m^-}{M^-}$ pour laquelle la satisfiabilité moyenne de (Q1) est faible. La zone d'échec est observée en affichant la satisfiabilité moyenne de (Q1) dans le plan $\left(\frac{m^+}{M^+}, \frac{m^-}{M^-}\right)$. La satisfiabilité est affichée en noir ou blanc lorsqu'elle est respectivement de 0 ou 100%.

5.2 Confirmation des résultats théoriques

Avec les paramètres d'ordre par défaut (tableau 5.1), la satisfiabilité moyenne évolue comme le montre la figure 5.1. On retrouve comme prévu une zone, autour de $\frac{m^+}{M^+} = \frac{m^-}{M^-}$, pour laquelle la satisfiabilité moyenne est proche de 0.

Cette section étudie la sensibilité de la zone d'échec par rapport aux différents paramètres d'ordre.

[1] La résolution d'un problème de satisfaction de contraintes (Q1) à l'aide de GLPK nécessite en moyenne 16 secondes, sur un Pentium IV avec une fréquence de 3GHz et une mémoire vive de 1Go.

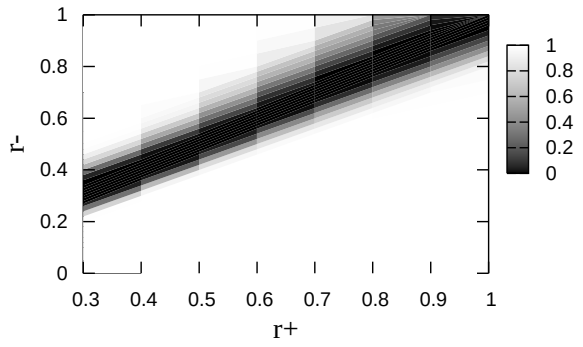


FIG. 5.1 – Fraction de problème de satisfaction de contraintes (Q1) satisfiables, sur 40 tirages indépendants, en fonction de $r+ = \frac{m^+}{M^+}$ et $r- = \frac{m^-}{M^-}$. Les valeurs des paramètres utilisés pour générer les problèmes sont indiquées dans le tableau 5.1.

5.2.1 Effet du nombre de concepts élémentaires « manqués » par un exemple négatif

Les premières expériences concernent l'effet du nombre P_m de concepts élémentaires manqués par un exemple négatif. La figure 5.2 montre les résultats obtenus en fixant les paramètres d'ordre aux valeurs par défaut (tableau 5.1), excepté P_m qui prend les valeurs 10, 20 et 25.

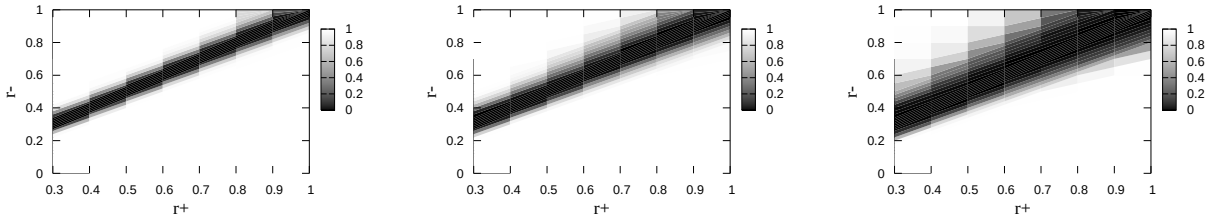
Comme prédit par la section 4.3.2, la zone d'échec augmente lorsque P_m augmente. Pourtant P_m n'influence pas l'espérance de la valeur prise par le noyau-somme K, et par suite ne modifie pas la différence entre la représentation moyenne associée à un exemple positif et la représentation moyenne associée à un exemple négatif. Mais la variance de ces deux représentations augmente avec P_m, donc plus P_m est grand, plus il est difficile de séparer les exemples positifs des exemples négatifs, et donc plus la zone d'échec s'étend.

5.2.2 Effet des tailles des ensembles d'entraînement et de test

Comme on peut s'y attendre, la zone d'échec diminue lorsqu'on augmente le nombre n d'exemples d'entraînement (figure 5.3). La diminution de la zone d'échec s'explique de deux façon complémentaires, sous réserve du fait que $\frac{m^+}{M^+}$ et $\frac{m^-}{M^-}$ diffèrent.

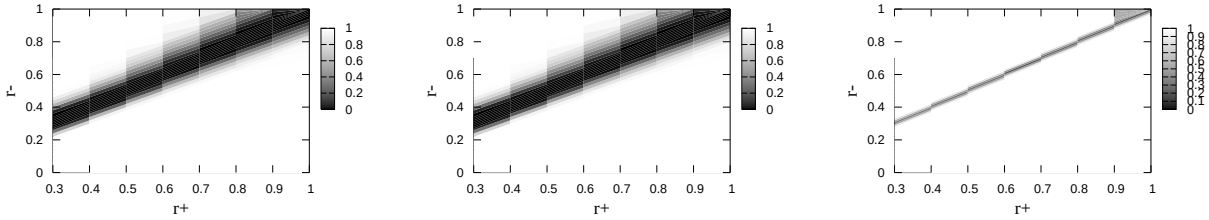
Tout d'abord, augmenter le nombre d'exemples d'entraînement permet d'accroître la distance dans la représentation $\mathcal{R}_\mathcal{E}$ entre la moyenne des exemples positifs et la moyenne des exemples négatifs. Coordonnée par coordonnée, n ne modifie pas la différence de valeur entre les exemples positifs et les exemples négatifs ; en revanche, la distance entre la moyenne des exemples positifs et la moyenne des exemples négatifs est proportionnelle à $\sqrt{n}$ puisqu'elle fait intervenir les n coordonnées.

De plus, plus l'ensemble d'entraînement est important, plus il a de chance de contenir un exemple $x^\star$ qui possède le même nombre d'instances dans chaque concept élémentaire (ou un exemple qui s'en approche). L'avantage d'un tel exemple est que le noyau-somme entre cet exemple et tout autre



(a) $P_m = 10$ (b) $P_m = 20$ (c) $P_m = 25$

FIG. 5.2 – **Influence de P_m**. Fraction de problèmes de satisfaction de contraintes (Q1) satisfiables, sur 40 tirages indépendants, en fonction de $r+ = \frac{m^+}{M^+}$ et $r- = \frac{m^-}{M^-}$. Les valeurs par défaut des paramètres utilisés pour générer les problèmes sont indiquées dans le tableau 5.1, excepté pour le nombre P_m de concepts élémentaires manqués par les exemples négatifs qui varie dans $\{10, 20, 25\}$.



(a) $n = 20$ (10 +, 10 −) (b) $n = 60$ (30 +, 30 −) (c) $n = 180$ (90 +, 90 −)

FIG. 5.3 – **Influence de n**. Fraction de problèmes de satisfaction de contraintes (Q1) satisfiables, sur 40 tirages indépendants, en fonction de $r+ = \frac{m^+}{M^+}$ et $r- = \frac{m^-}{M^-}$. Les valeurs par défaut des paramètres utilisés pour générer les problèmes sont indiquées dans le tableau 5.1, excepté pour le nombre n d'exemples d'entraînement qui varie dans $\{20, 60, 180\}$.

exemple x est constant conditionnellement à la classe de x. Plus précisément

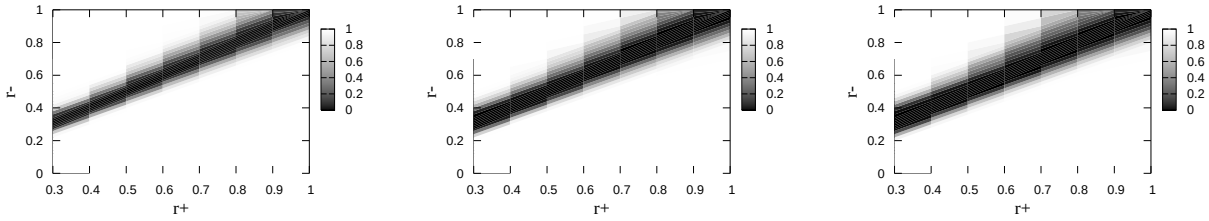
$$K(x^\star, x) = \begin{cases} \left(\frac{m^+}{M^+}\right)^2 \frac{\mu_1 - \mu_0}{P} + \mu_0 & \text{si } x \text{ est positif} \\ \frac{m^+}{M^+} \frac{m^-}{M^-} \frac{\mu_1 - \mu_0}{P} + \mu_0 & \text{si } x \text{ est négatif} \end{cases}$$

Ainsi, une fois que $x^\star$ fait partie de l'ensemble d'entraînement, la classe d'un exemple x peut être déterminée en calculant uniquement $K(x^\star, x)$.

Un autre comportement attendu est le fait que la zone d'échec croît avec le nombre d'exemples de test (figure 5.4). En effet chaque exemple de test ajoute une contrainte à (Q1) ; la probabilité que (Q1) soit satisfiable diminue donc avec le nombre n' d'exemples de test.

5.2.3 Effet d'un bruit sur le nombre d'instances pertinentes par exemple

Afin d'examiner l'impact de m^+ et m^-, des expériences complémentaires ont été réalisées en faisant varier le nombre d'instances pertinentes pour les exemples d'entraînement ou de test.



(a) $n' = 100$ (50 +, 50 −) (b) $n' = 200$ (100 +, 100 −) (c) $n' = 400$ (200 +, 200 −)

FIG. 5.4 – **Influence de n'**. Fraction de problèmes de satisfaction de contraintes (Q1) satisfiables, sur 40 tirages indépendants, en fonction de $r+ = \frac{m^+}{M^+}$ et $r- = \frac{m^-}{M^-}$. Les valeurs par défaut des paramètres utilisés pour générer les problèmes sont indiquées dans le tableau 5.1, excepté pour le nombre n' d'exemples de test qui varie dans $\{100, 200, 400\}$.

Bruit pour les exemples d'entraînement Tout d'abord, le nombre d'instances pertinentes pour chaque exemple d'entraînement positif (respectivement négatif) est tiré uniformément dans l'intervalle $[|m^+ - \Delta, m^+ + \Delta|]$ (respectivement $[|m^- - \Delta, m^- + \Delta|]$), avec Δ variant dans $\{0, 5, 10\}$, alors que le nombre d'instances pertinentes pour les exemples de test reste constant.

Quand Δ augmente, la taille de la zone d'échec reste quasiment constante (figure 5.5). Ainsi, même si le bruit sur le nombre d'instances pertinentes pour les exemples d'entraînement permet d'obtenir une représentation $\mathcal{R}_\mathcal{E}$ avec des attributs plus variés, celà ne permet pas pour autant de discriminer plus facilement les exemples.

Bruit sur les exemples de test La figure 5.6 montre les résultats obtenus lorsque le nombre d'instances pertinentes pour chaque exemple de test positif (respectivement négatif) est tiré uniformément dans l'intervalle $[|m^+ - \Delta, m^+ + \Delta|]$ (respectivement $[|m^- - \Delta, m^- + \Delta|]$), avec Δ variant dans $\{0, 5, 10\}$, alors que le nombre d'instances pertinentes pour les exemples d'entraînement reste constant.

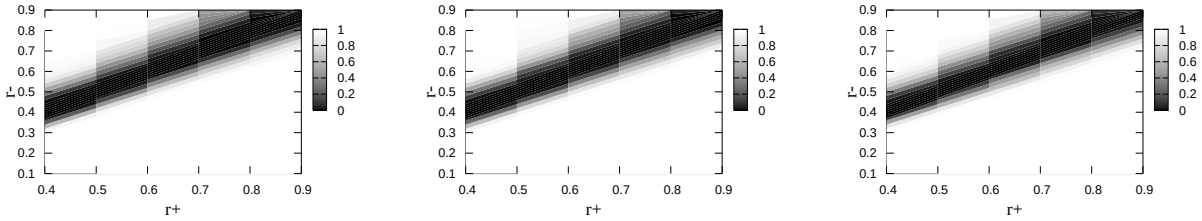
Cette fois-ci, la surface de la zone d'échec augmente avec Δ. En effet, lorsque δ augmente, il en va de même pour la variance de la représentation des exemples de test, ce qui rend plus difficile leur séparation en un groupe d'exemples négatifs et un groupe d'exemples positifs.

Bruit sur tous les exemples La figure 5.7 montre les résultats obtenus lorsque le nombre d'instances pertinentes varie à la fois pour les exemples d'entraînement et pour les exemples de test. C'est alors l'effet induit par le bruit sur les exemples de test qui l'emporte : la zone d'échec croît avec Δ.

5.2.4 Effet de l'univers

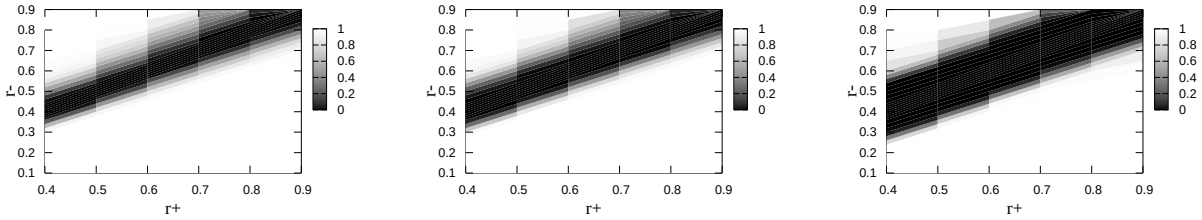
Jusqu'à présent, nous avons supposé que les instances sont soit tirées dans un concept élémentaire, soit tirées uniformément en dehors du concept-cible. On parlera d'univers *uniforme*.

Cette section décrit comment les résultats changent quand les instances négatives sont au contraire tirées dans des boules définissant un concept-univers (section 4.2). Il semble en effet



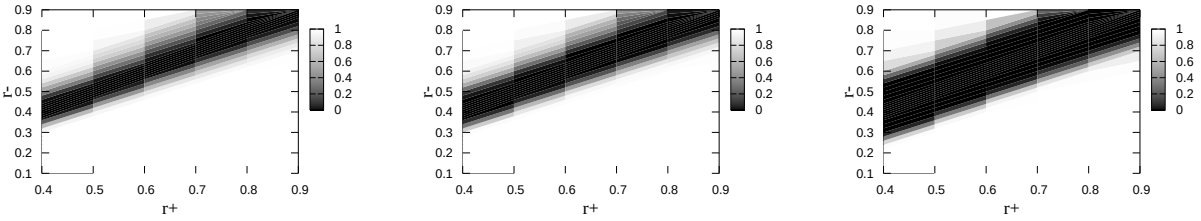
(a) $\Delta = 0$ (b) $\Delta = 5$ (c) $\Delta = 10$

FIG. 5.5 – **Influence d'un bruit sur m^+ et m^- pour les exemples d'entraînement**. Fraction de problèmes de satisfaction de contraintes (Q1) satisfiables, sur 40 tirages indépendants, en fonction de $r+ = \frac{m^+}{M^+}$ et $r- = \frac{m^-}{M^-}$. Les valeurs par défaut des paramètres utilisés pour générer les problèmes sont indiquées dans le tableau 5.1. Pour les exemples d'entraînement, m^+ (respectivement m^-) n'est pas constant, mais choisi uniformément pour chaque exemple dans l'intervalle $[|m^+ - \Delta, m^+ + \Delta|]$ (respectivement $[|m^- - \Delta, m^- + \Delta|]$) avec Δ variant dans $\{0, 5, 10\}$.



(a) $\Delta = 0$ (b) $\Delta = 5$ (c) $\Delta = 10$

FIG. 5.6 – **Influence d'un bruit sur m^+ et m^- pour les exemples de test**. Fraction de problèmes de satisfaction de contraintes (Q1) satisfiables, sur 40 tirages indépendants, en fonction de $r+ = \frac{m^+}{M^+}$ et $r- = \frac{m^-}{M^-}$. Les valeurs par défaut des paramètres utilisés pour générer les problèmes sont indiquées dans le tableau 5.1. Pour les exemples de test, m^+ (respectivement m^-) n'est pas constant, mais choisi uniformément pour chaque exemple dans l'intervalle $[|m^+ - \Delta, m^+ + \Delta|]$ (respectivement $[|m^- - \Delta, m^- + \Delta|]$) avec Δ variant dans $\{0, 5, 10\}$.



(a) $\Delta = 0$ (b) $\Delta = 5$ (c) $\Delta = 10$

FIG. 5.7 – **Influence d'un bruit sur m^+ et m^-.** Fraction de problèmes de satisfaction de contraintes (Q1) satisfiables, sur 40 tirages indépendants, en fonction de $r+ = \frac{m^+}{M^+}$ et $r- = \frac{m^-}{M^-}$. Les valeurs par défaut des paramètres utilisés pour générer les problèmes sont indiquées dans le tableau 5.1. m^+ (respectivement m^-) n'est pas constant, mais choisi uniformément pour chaque exemple dans l'intervalle $[|m^+ - \Delta, m^+ + \Delta|]$ (respectivement $[|m^- - \Delta, m^- + \Delta|]$) avec Δ variant dans $\{0, 5, 10\}$.

concevable qu'une distribution non-uniforme des instances non-pertinentes entraîne des résultats différents lors de l'apprentissage fondé sur le noyau-somme.

Effet de la taille de l'univers Nous supposons ici que le concept-univers est composé de Q boules, et nous observons l'effet de la valeur de Q sur la qualité de la représentation $\mathcal{R}_\mathcal{E}$.

Tout d'abord, on remarque que la zone d'échec observée lorsque Q est fixé à mille (figure 5.8(c)) est similaire à la zone d'échec observée avec l'univers uniforme (figure 5.1). Avec un univers d'une taille intermédiaire de $Q = 30$ boules, la zone d'échec est un peu plus importante lorsque m^+ et m^- sont proches de cinquante (figure 5.8(b)). Enfin, avec un petit univers composé de $Q = 5$ boules, la zone d'échec est très fine (figure 5.8(a)).

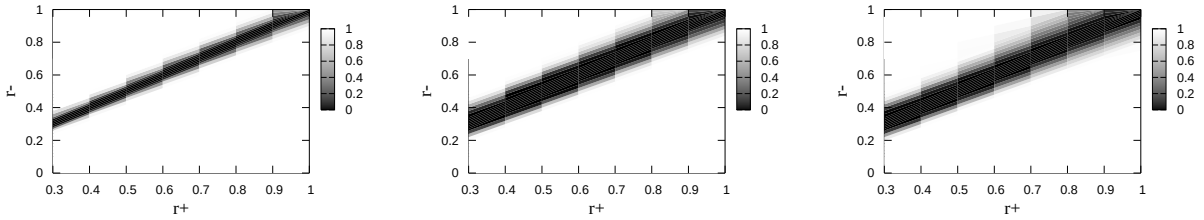
Ce comportement non-monotone est conforme à l'analyse présentée en section 4.3.3. Le ratio entre $|\Delta_K|$ avec l'univers uniforme et $|\Delta_K|$ en présence d'un univers composé de mille boules est proche de 1 lorsque $\frac{m^+}{M^+}$ ou $\frac{m^-}{M^-}$ est plus grand que $0,2$ (figure 4.3). Un univers composé de mille boules a donc un comportement similaire à l'univers uniforme.

Pour un univers avec trente boules, ce ratio est proche de 1 ou supérieur à 1, excepté au voisinage de $\frac{m^+}{M^+} = \frac{m^-}{M^-} = 0,5$ (figure 4.3). D'où le fait que la zone d'échec soit un peu plus importante lorsque m^+ et m^- sont proches de cinquante.

Enfin pour un univers composé de cinq boules, le ratio est supérieur à 1, excepté au voisinage de $\frac{m^+}{M^+} = \frac{m^-}{M^-} = 0,85$ (figure 4.3). D'où la finesse de la zone d'échec.

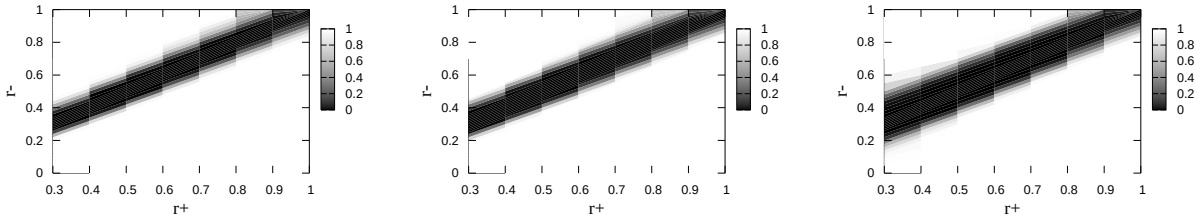
Effet du nombre de boules de l'univers « manquées » par un exemple positif Le nombre Q_m de boules de l'univers manquées par un exemple positif et le nombre P_m de concepts élémentaires manqués par un exemple négatif ont un effet similaire. Ils n'interviennent pas sur la moyenne de la représentation $\Phi(x)$ d'un exemple x, mais ils modifient sa variance : quand Q_m croît, la variance de $\Phi(x)$ en fait de même, et par suite moins de problèmes de satisfaction de contraintes sont satisfiables.

Toutefois, l'effet de Q_m est plus important lorsque les ratios $\frac{m^+}{M^+}$ et $\frac{m^-}{M^-}$ sont petits (figure 5.9). Ce comportement est raisonnable puisque Q_m agit sur les $M^+ - m^+$ ou $M^- - m^-$ instances de



(a) $Q = 5$ (b) $Q = 30$ (c) $Q = 1\,000$

FIG. 5.8 – **Influence de Q.** Fraction de problèmes de satisfaction de contraintes (Q1) satisfiables, sur 40 tirages indépendants, en fonction de $r+ = \frac{m^+}{M^+}$ et $r- = \frac{m^-}{M^-}$. Les valeurs par défaut des paramètres utilisés pour générer les problèmes sont indiquées dans le tableau 5.1. Les instances non pertinentes sont tirées dans un univers composé de Q boules, avec Q variant dans $\{5, 30, 1\,000\}$.



(a) $Q_m = 0$ (b) $Q_m = 15$ (c) $Q_m = 25$

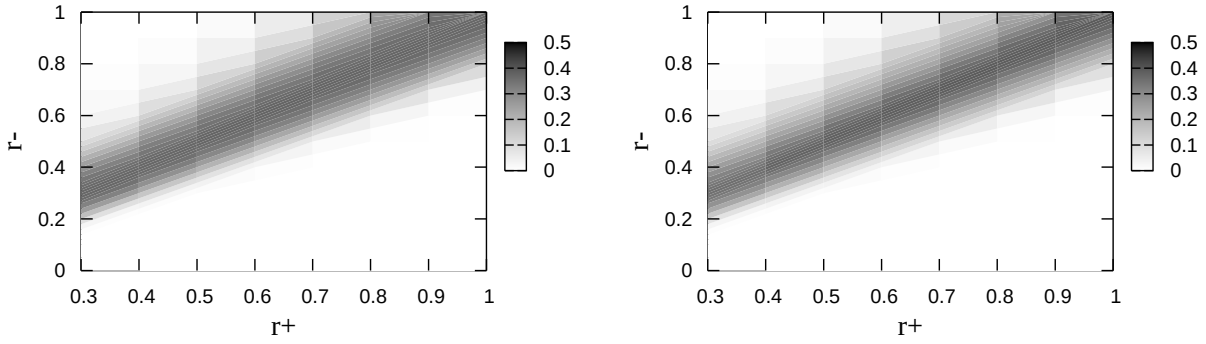
FIG. 5.9 – **Influence de Q_m.** Fraction de problèmes de satisfaction de contraintes (Q1) satisfiables, sur 40 tirages indépendants, en fonction de $r+ = \frac{m^+}{M^+}$ et $r- = \frac{m^-}{M^-}$. Les valeurs par défaut des paramètres utilisés pour générer les problèmes sont indiquées dans le tableau 5.1. Les instances non pertinentes sont tirées dans un univers composé de $Q = 30$ boules. Chaque exemple positif manque Q_m boules de l'univers, avec Q_m variant dans $\{0, 15, 25\}$.

chaque exemple qui sont tirées dans l'univers. Q_m a donc plus d'effet lorsque $M^+ - m^+$ ou $M^- - m^-$ est grand.

5.3 Erreur en généralisation

Dans la section précédente, le critère de puissance de la représentation a permis de visualiser en pratique la zone d'échec des noyaux-somme. Ce critère est lié à une borne inférieure sur l'erreur en généralisation, mais, étant donné le petit nombre de problèmes traités, cette borne inférieure reste peu contraignante : un taux de non-satifiabilité de 100% sur quarantes problèmes ne permet de conclure qu'à une erreur en généralisation supérieure à 0,8% avec un niveau de confiance de 95%.

Afin de vérifier que cette borne permet d'identifier les zones difficiles, les ensembles d'entraînement et de test générés précédemment sont réutilisés pour estimer, cette fois-ci, l'erreur en généralisation d'une machine à vecteurs de support. Pour chaque problème $(\mathcal{E}, \mathcal{T})$, où $\mathcal{E}$ est un ensemble d'entraînement et $\mathcal{T}$ un ensemble de test, une machine à vecteurs de support est entraînée sur $\mathcal{E}$ et testée sur $\mathcal{T}$. Pour chaque jeu de paramètres d'ordre, l'erreur en généralisation est estimée par



(a) $C = 100$ (b) $C = 1\ 000\ 000$

FIG. 5.10 – Erreur en généralisation, estimée sur 40 tirages indépendants, en fonction de $r+ = \frac{m^+}{M^+}$ et $r- = \frac{m^-}{M^-}$. Les valeurs des paramètres utilisés pour générer les problèmes sont indiquées dans le tableau 5.1.

l'erreur en test moyenne obtenue sur les quarante problèmes générés.

Les machines à vecteurs de support sont entraînées à l'aide de SVMTorch [Collobert 02] en utilisant les paramètres par défaut, excepté le paramètre C controlant le compromis entre la marge et l'erreur empirique[2]. La figure 5.10 montre l'évolution de l'erreur en généralisation en fonction de $\frac{m^+}{M^+}$ et $\frac{m^-}{M^-}$ en utilisant les valeurs par défaut des paramètres d'ordre (tableau 5.1) ; l'erreur en généralisation est affichée en noir ou blanc lorsqu'elle est respectivement de 50 ou 0%.

Que ce soit avec $C = 10^2$ ou $C = 10^6$, les résultats sont similaires et confirment que l'erreur des machines à vecteurs de support croît avec la non-satisfiabilité du problème (Q1). Alors que la proportion de problèmes non-satisfiables passe rapidement de 0 à 100% (figure 5.1), l'erreur en généralisation augmente de façon plus continue (figure 5.10), mais significativement ; quand la proportion de problèmes non-satisfiables est supérieure à 80%, l'erreur en généralisation dépasse 30%.

5.4 Résumé

Ce chapitre confirme et affine l'analyse théorique du chapitre 4, portant sur la satisfiabilité du problème (Q1), soit l'existance d'une solution au sens du porblème induit par le noyau-somme :

- lorsque $\frac{m^+}{M^+}$ et $\frac{m^-}{M^-}$ sont proches, le problème de satisfaction de contrainte (Q1) est difficilement satisfiable ;
- plus les exemples négatifs manquent de concepts élémentaires, plus la zone d'échec s'étend ;
- l'univers modifie la zone d'échec de façon variée suivant sa taille. Avec un petit univers, la zone d'échec diminue ; avec un univers de taille moyenne, la zone d'échec s'élargit autour de $\frac{m^+}{M^+} = \frac{m^-}{M^-} = \frac{1}{P+Q}$; avec un univers de grande taille, on retrouve les résultats obtenus sans

[2]L'entraînement d'un SVM nécessite en moyenne 25 secondes, sur un Pentium IV avec une fréquence de 3GHz et une mémoire vive de 1Go.

univers.

Ces résultats proviennent en grande partie du fait qu'une machine à vecteurs de support utilisant le noyau-somme tend à classer les exemples d'après leur proportion d'instances pertinentes et non pas d'après la répartition de ces instances. Il n'y a pas de magie : le noyau-somme n'est pas en mesure de reconstruire une information existentielle à partir d'une information moyenne.

Cependant, le noyau-somme n'est définitivement mis en échec que lorsque $\frac{m^+}{M^+} = \frac{m^-}{M^-}$. Les différents paramètres d'ordres peuvent rendre plus difficile la séparation entre exemples positifs et exemples négatifs lorsque $\frac{m^+}{M^+}$ et $\frac{m^-}{M^-}$ sont proches ; mais l'augmentation de la taille du problème d'apprentissage est comme on pouvait s'y attendre un facteur positif. En revanche, l'augmentation de la dispersion des caractéristiques des exemples (figure 5.7) est un facteur négatif.

Chapitre 6

Discussion et perspectives

Cette première partie a rappelé les difficultés et les limites spécifiques à l'apprentissage relationel, et s'est posé la question de savoir dans quelle mesure l'astuce des noyaux permettait de les résoudre.

Résultats Dans cette partie de la thèse nous nous sommes intéressés au noyau-somme, qui est utilisé dans le cadre des problèmes à instances multiples. L'intérêt de ces problèmes est qu'ils sont un intermédiaire entre le problèmes propositionnels et relationnels.

Cette étude s'est concentré sur une famille de concepts existentiels conjonctifs. Nous avons montré théoriquement et expérimentalement que, pour cette famille de concepts, le noyau-somme n'est pas en mesure de retrouver le concept-cible lorsque la proportion d'instances pertinentes est la même pour les exemples positifs que pour les exemples négatifs.

Cette zone d'échec du noyau-somme est restreinte, mais elle met en évidence la limite du noyau-somme : au lieu de reconstruire un concept relationnel, le noyau-somme classe les exemples en fonction de propriétés moyennes. Dans notre étude par exemple, le noyau-somme ne différencie pas les exemples d'après la distribution de leur instances, mais simplement d'après le nombre d'instances pertinentes. Au passage, le noyau-somme ne fait pas non-plus la différence entre les instances pertinentes ; ce noyau ne fait que découper l'espace des instance en deux parties : l'union des concepts élémentaires et « le reste ».

La mise en évidence expérimentale de la zone d'échec a été permise grâce à un nouveau critère directement relié à une borne inférieure sur l'erreur en généralisation. Ce critère permet de tester directement la qualité de la représentation induite par le noyau-somme. Ainsi, au lieu de regarder l'erreur en généralisation pour quelques algorithmes, on peut directement observer ce qu'aurait pu faire au mieux n'importe quel algorithme.

Pour aller plus loin Une autre approche pour s'attaquer aux problèmes à instances multiples utilise un algorithme en deux étapes [Chen 04, Bi 05] : une première étape pour déterminer les zones de l'espace des instances où se trouvent les instances pertinentes, et une deuxième étape utilisant un algorithme standard pour apprendre une hypothèse. L'algorithme utilisé dans la deuxième étape

ne travaille pas sur la représentation des exemples sous forme d'instances multiples, mais sur une propositionnalisation des exemples.

Plus précisément, à la première étape l'algorithme recherche des instances $I_1, \ldots, I_L$ représentant les centres des zones pertinentes (les instances sont des vecteurs de $\mathbb{R}^d$). Puis à chaque exemple x est associé la représentation

$$\Psi(x) = (d(I_1, x), \ldots, d(I_L, x))$$

où pour toute instance I_ℓ représentant une zone pertinente, $d(I_\ell, x)$ est la distance de I_ℓ aux instances de x :

$$d(I_\ell, x) = \min_{I \in x} \|I - I_j\|_2$$

Enfin, un algorithme d'apprentissage classique est utilisé pour séparer les exemples positifs des exemples négatif en se fondant sur la représentation Ψ.

[Chen 06] obtient de bons résultats en utilisant toutes les instances de tous les exemples pour construire Ψ, au lieu de ne sélectionner que quelques instances. Le choix des instances fournissant le plus d'information est laissé à un algorithme de sélection d'attributs. Concrètement, [Chen 06] utilise une machine à vecteurs de support régularisée en norme 1 mais d'autres algorithmes de sélection d'attributs peuvent être envisagés.

La représentation utilisant tous les exemples nous semble la plus prometteuse, c'est pourquoi dans la deuxième partie de cette thèse nous nous intéresserons à la sélection d'attributs.

Deuxième partie

Sélection d'attributs

La sélection d'attributs, un des objectifs-clé en apprentissage automatique, est un problème d'optimisation combinatoire dont l'objectif est de minimiser l'erreur en généralisation.

La sélection d'attributs est abordée selon trois approches principales : les approches filtre (filter), les approches enveloppe (wrapper) et les approches embarquées (embedded). Les approches filtre se fondent sur des critères statistiques pour ordonner les attributs en fonction du problème de classification. La principale limitation des approches filtre est de ne pas tenir suffisamment compte des dépendances entre attributs. Les approches enveloppe adressent directement le problème combinatoire. Ces approches explorent directement l'ensemble de sous-parties de l'ensemble d'attributs en associant à chaque sous-ensemble d'attributs candidat une estimation de son erreur en généralisation. Les approches embarquées exploitent le modèle appris et/ou incorporent des critères de parcimonie pendant la phase d'apprentissage.

La sélection d'attributs passe par la détection (et la suppression) des attributs non-pertinents ou redondants. La difficulté principale est que l'intérêt d'un attribut dépend des autres attributs sélectionnés. Par exemple, un attribut redondant n'est redondant que par rapport à d'autres attributs, ou, à l'opposé, certains attributs ne sont informatifs qu'une fois utilisés avec d'autres attributs. Pour trouver la solution optimale, un algorithme de sélection d'attributs se doit donc de tester l'intérêt d'un attribut dans le contexte d'autres attributs. Mais si ce test se fait dans le contexte de tous les attributs, il sera perturbé par les attributs non-pertinents ou redondants. Ainsi, les approches de type filtre ou les approches embarquées ne peuvent trouver le meilleur sous-ensemble d'attributs car elles se limitent à tester les attributs soit indépendamment les uns des autres, soit dans le contexte de tous les attributs. Au contraire, les méthodes enveloppe peuvent trouver le sous-ensemble d'attributs optimal, mais au prix d'un temps de calcul élevé puisque qu'il leur faut potentiellement tester tous les sous-ensembles d'attributs.

Cette partie de la thèse formalise la sélection d'attributs dans sa version enveloppe comme un problème d'apprentissage par renforcement. Cette formalisation conduit de façon certaine à une politique optimale mais non calculable dans un temps raisonnable. Un approximation de cette politique est obtenue en considérant ce problème d'apprentissage par renforcement et en appliquant le cadre UCT (pour Upper Confidence bound applied to Trees), qui est une approche robuste pour l'optimisation sous incertitude.

Cette partie est organisée comme suit. Les chapitre 7 et 8 passent en revue la sélection d'attributs, UCT, et le problème du bandit manchot, dont UCT est une extension au cas des arbres. Le chapitre 9 décrit la réécriture du problème de sélection d'attributs en un problème d'apprentissage par renforcement, ainsi que les adaptations apportées à UCT pour lui permettre d'approcher la politique optimale de ce problème d'apprentissage par renforcement. Enfin, le chapitre 10 présente les résultats obtenus par l'algorithme correspondant (FUSE pour *Feature Uct SElection*) sur des bases de données de référence, illustrant ainsi la capacité de l'algorithme à être arrêté à tout instant ainsi que ses bons résultats.

Chapitre 7

État de l'art : Sélection d'attributs

Le problème abordé dans cette partie de la thèse est la sélection d'attributs. Ce chapitre précise tout d'abord les questions posées par la sélection d'attributs et les grandes stratégies développées dans la littérature pour répondre à ces questions.

7.1 Position du problème

Dans ce chapitre, nous nous focaliserons sur la sélection d'attributs dans le contexte de l'apprentissage supervisé propositionnel [Guyon 03b]. Etant donné un ensemble d'apprentissage $\mathcal{E} = \{(x_i, y_i), x_i \in \mathcal{X}, y_i \in \mathcal{Y}, i = 1 \dots n\}$, soit $\mathcal{F} = \{f_1, \dots, f_d\}$ l'ensemble des attributs descriptifs disponibles. Par définition, l'espace des instances $\mathcal{X}$ correspond au produit cartésien des domaines $(\mathcal{X}_i)_{i \in 1,\dots,d}$ des attributs de $\mathcal{F}$. Dans le cas où $\mathcal{F}$ comprend d attributs numériques, on a ainsi $\mathcal{X} = \mathbb{R}^d$.

Un algorithme d'apprentissage $\mathcal{A}$ apprend à partir de $\mathcal{E}$ une hypothèse $h : \mathcal{X} \mapsto \mathcal{Y}$, et la qualité de cette hypothèse est classiquement donnée par l'espérance de son coût d'erreur. Soit $\ell : \mathcal{Y} \times \mathcal{Y} \mapsto \mathbb{R}^+$ la fonction de coût, où $\ell(h(x), y)$ représente le coût de labeller un exemple x par $h(x)$ au lieu du vrai label y. Alors, l'erreur en généralisation de l'hypothèse h est donnée par :

$$\mathbf{Err}\,(h) = \mathbb{E}_{(x,y)\sim P_{\mathcal{XY}}}\left[\ell\left(h(x), y\right)\right] \tag{7.1}$$

L'erreur en généralisation est bien entendu hors de notre portée, dans la mesure où elle demande la connaissance de la distribution jointe $P_{\mathcal{XY}}$ des données. Un algorithme d'apprentissage $\mathcal{A}$ cherche ainsi une hypothèse h effectuant un bon compromis entre l'erreur empirique mesurée sur les données d'apprentissage, qui doit être minimisée

$$\mathbb{E}_{(x,y)\in\mathcal{E}}\left[\ell\left(h(x), y\right)\right] = \frac{1}{n}\sum_{i=1}^{n} \ell\left(h(x_i), y_i\right)$$

et la variance, mesurant l'écart possible entre l'erreur empirique et l'erreur en généralisation sur la classe d'hypothèses H considérée. La théorie de la mesure permet de borner cet écart en fonction

de la qualité de l'estimation empirique (dépendant du nombre n d'exemples sur lesquels calculer l'estimation empirique) et de la régularité de l'espace H considéré (mesurée par la dimension de Vapnik-Cervonenkis de H) ; cet écart correspond au risque de sur-apprentissage (overfitting). Ainsi une approche classique de l'apprentissage, l'apprentissage statistique [Vapnik 98], consiste à minimiser une borne supérieure de l'erreur en généralisation, dépendant de l'erreur empirique, du nombre d'exemples, et de la dimension de Vapnik-Cervonenkis de l'espace des hypothèses. Plus généralement, l'apprentissage cherche simultanément à minimiser l'erreur empirique, et à éviter le sur-apprentissage en optimisant un terme dit de régularisation.

La sélection d'attributs, remplaçant l'ensemble $\mathcal{F}$ initial par un sous-ensemble d'attributs $F \subset \mathcal{F}$, a comme objectif premier de réduire l'erreur en généralisation de l'hypothèse apprise. Cette réduction peut intervenir à la fois sur le risque de sur-apprentissage et sur l'erreur empirique. En ce qui concerne le risque de sur-apprentissage tout d'abord, il est clair que le fait de diminuer l'ensemble des attributs diminue la richesse de l'ensemble H des hypothèses considéré, toutes choses égales par ailleurs ; il réduit sa dimension de Vapnik-Cervonenkis et donc la borne supérieure sur l'erreur en généralisation. Mais même en ce qui concerne l'erreur empirique, la réduction de l'espace de recherche H peut faciliter son optimisation pour des raisons algorithmiques ou numériques.

7.1.1 Discussion

Nous nous concentrerons dans la suite sur la sélection d'attributs en vue de la minimisation de l'erreur en généralisation. Il convient néanmoins de signaler que la sélection d'attributs peut également viser d'autres objectifs :

- Réduire la complexité de l'apprentissage[1].
- Diminuer le coût d'usage de l'hypothèse apprise, soit en complexité algorithmique et temps de traitement, soit en coût d'expertise et/ou financier puisqu'il devient nécessaire de renseigner moins d'attributs.
- Augmenter la lisibilité de l'hypothèse et faciliter l'interaction entre l'expert et l'algorithme d'apprentissage.

Dans la suite, la sélection d'attributs est abordée comme un problème de minimisation combinatoire, donné par :

$$\text{Trouver } F^* = \underset{F \subseteq \mathcal{F}}{\operatorname{argmin}} \ \mathbf{Err}\left(\mathcal{A}\left(F, \mathcal{E}\right)\right) \tag{7.2}$$

où $\mathcal{A}(F, \mathcal{E})$ est l'hypothèse apprise par l'algorithme $\mathcal{A}$ à partir de l'ensemble d'entraînement $\mathcal{E}$ et en utilisant uniquement les attributs contenus dans F.

La difficulté de ce problème est double. En premier lieu, il s'agit (équation 7.2) d'un problème d'optimisation combinatoire sur $\{0,1\}^{|\mathcal{F}|}$ noté P(F) ; l'espace de recherche est le treillis des parties de l'ensemble $\mathcal{F}$ des attributs. Il s'agit d'un problème de complexité exponentielle au pire cas.

En second lieu, la nature du critère à optimiser ne permet aucun élagage fondé de l'espace de recherche. En effet, la valeur de l'erreur en généralisation n'est pas connue et ne peut être qu'estimée. De plus, l'erreur en généralisation ne dispose pas de bonnes propriétés (monotonie) qui permettraient d'élaguer efficacement des parties de l'espace de recherche P(F).

En effet, l'objectif de la sélection d'attributs peut être intuitivement décrit comme le fait d'éliminer les attributs redondants et d'identifier les attributs (conditionnellement) pertinents :

[1] Notons que cet avantage qui est d'autant plus appréciable que la complexité de l'apprentissage est supra-linéaire par rapport au nombre d'attributs, est en général annulé par la complexité de la sélection d'attributs (section 7.2).

- un attribut f est dit redondant avec un ensemble F s'il n'apporte aucune information supplémentaire pour la classification par rapport à F, i.e. si f est conditionnellement indépendant de l'étiquette des exemples connaissant F ;
- un attribut f est conditionnellement pertinent relativement à un ensemble F si l'hypothèse apprise à partir de $F \cup \{f\}$ améliore sur celle apprise à partir de F. En particulier s'il est redondant avec F il n'est pas conditionnellement pertinent. Toutefois, le test de redondance est conceptuellement plus simple (moins bruité) que celui de pertinence conditionnelle.

La redondance ou la pertinence d'un attribut sont ainsi définies par rapport à un ensemble F donné, ce qui va dans le sens d'une exploration coûteuse de l'espace de recherche.

7.2 État de l'art

Comme expliqué dans la section précédente, la sélection d'attributs est un problème central en apprentissage automatique. En 1997, existent déjà des approches spécifiquement dédiées à la sélection d'attributs, même si les capacités de calcul de l'époque ne permettent de se confronter qu'à des bases de données comportant quelques dizaines ou quelques centaines d'attributs [Blum 97, Kohavi 97].

En 2003, Guyon et Elisseeff [Guyon 03b] proposent une vue d'ensemble de la sélection d'attributs. Il ressort alors que les méthodes de sélection d'attributs viennent de franchir une marche quantitative, en proposant de se confronter à des bases de données comportant plusieurs milliers d'attributs (données génomiques ([Guyon 02]) ou textuelles [Joachims 98]). Afin de comparer les différentes approches existentes, un concours de sélection d'attributs est proposé à l'occasion de la conférence NIPS 2003 [Guyon 04].

La sélection d'attributs reste un problème ouvert et la taille des bases de données ne cesse d'augmenter. Les données client stockées par les entreprises, par exemple, peuvent contenir plusieurs dizaines de milliers d'attributs [Feraud 10]. Fort de ce constat, un nouveau concours de sélection d'attributs a lieu à l'occasion de la conférence KDD 2009 [Guyon 10]. Plus de 450 équipes ont participé à ce concours, ce qui démontre bien l'intéret de la communauté pour la sélection d'attributs.

Les nombreuses approches pour sélectionner les attributs se regroupent de différentes manières. Du point de vue du type de corrélations entre attributs considérées, on distingue les algorithmes :

- *univariés* : chaque attribut est considéré indépendamment des autres. Il est donc impossible de détecter les redondances entre attributs ou les attributs conditionnellement pertinents ;
- *multi-variés linéaires* : on mesure la pertinence de groupes d'attributs lorsqu'ils sont utilisés sous forme de combinaisons linéaires ;
- *multi-variés non-linéaires* : on mesure la pertinence de groupes d'attributs quel que soit leur utilisation.

Plus on avance dans cette liste, plus les algorithmes mis en jeu sont compliqués et plus ces algorithmes nécessitent de temps de calcul.

Dans cette thèse nous donnons un aperçu des approches de sélection d'attribut en utilisant un autre regroupement traditionnel en trois grandes familles : les méthodes *filtre* (filter), les méthodes *enveloppe* (wrapper) et les méthodes *embarquées* (embedded). Ce regroupement correspond à une schématisation du traitement des données.

- Les méthodes filtre agissent comme un filtre en ne fournissant qu'une partie des données à l'algorithme d'apprentissage.

- Les méthodes enveloppe proposent successivement plusieurs sous-ensembles d'attributs jusqu'à trouver le meilleur.
- Les méthodes embarquées considèrent simultanément le problème de l'apprentissage et le problème de la sélection d'attributs.

7.2.1 Méthodes filtre

Les méthodes filtre associent un score de pertinence à chaque attribut ; ce score induit ainsi un ordre total $f_{i_1} > \ldots > f_{i_d}$ sur l'ensemble des attributs, qui permettra ensuite de se limiter à explorer itérativement les sous-ensembles $\{f_{i_1}, \ldots f_{i_j}\}$ pour $j = 1, \ldots, d$, soit un nombre linéaire de tests en fonction du nombre d'attributs.

Parmi les scores mesurant la corrélation entre un attribut et l'étiquette des exemples, on trouve de nombreux score univariés, i.e. des score qui s'intéressent à chaque attribut indépendamment des autres.

- Le *test de Student* (T-test) est un test pour données binaires ($\mathcal{Y} = \{+1, -1\}$) [Furey 00, Hastie 01]. Ce test suppose que les attributs sont tirés selon une gaussienne et compare, pour un attribut f_i fixé, les moyennes et variances des exemples positifs ($\hat{\mu}_i^+$ et $\hat{\sigma}^+$) et négatifs ($\hat{\mu}_i^-$ et $\hat{\sigma}^-$). Si l'attribut f_i est pertinent, les deux moyennes diffèrent comparé à leur variance. Formellement le score associé à f_i est la T-statistique

$$t(f_i) = \frac{|\hat{\mu}_i^+ - \hat{\mu}_i^-|}{s\sqrt{\frac{1}{n^+} + \frac{1}{n^-}}}$$

  avec

$$s(f_i) = \sqrt{\frac{(n^+ - 1)\hat{\sigma}^+ + (n^- - 1)\hat{\sigma}^-}{n^+ + n^- - 2}}$$

  et n^+ (respectivement n^-) le nombre d'exemples positifs (respectivement négatifs). Plus la statistique $t(f_i)$ est élevée, plus l'attribut f_i est pertinent.
- L'*analyse de variance* (ANOVA pour *analysis of variance*) étend le test de Student aux cas avec plus de deux groupes et permet donc d'envisager une sélection d'attributs pour des problèmes de classification à plus de deux classes.
- Le *test du* χ^2 est un autre test statistique utilisé pour ordonner les attributs [Forman 03]. Ce test compare la distribution observée de $(f_i(x), y)$ avec la distribution correspondante en supposant que ces deux variables sont indépendantes. Lorsque $\mathcal{X}$ et $\mathcal{Y}$ sont discrets, le test du χ^2 se fonde sur la statistique :

$$\mathrm{CHI}(f_i) = \sum_{a \in \mathcal{X}_i} \sum_{b \in \mathcal{Y}} \frac{(P(f_i(x) = a, y = b) - P(f_i(x) = a)P(y = b))^2}{P(f_i(x) = a)P(y = b)}$$

  où $P(f_i(x) = a)$ et $P(y = b)$ sont les probabilités empiriques de $f_i(x)$ et y, et $P(f_i(x) = a, y = b)$ est la probabilité jointe empirique. Lorsqu'un attribut f_i n'est pas pertinent, $CHI(f_i)$ suit une loi du χ^2, qui est tabulée. Il est donc possible d'associer une probabilité de rejet à chaque attribut. Plus cette probabilité est importante, plus un attribut est considéré comme pertinent.
- L'*information mutuelle* entre un attribut et la classe est un indicateur de la pertinence de cet attribut. L'information mutuelle se définit formellement par :

$$\mathrm{IM}(f_i) = \int_{\mathcal{X}} \int_{\mathcal{Y}} p(f_i(x), y) \log \frac{p(f_i(x), y)}{p(f_i(x))p(y)} dxdy$$

où $p(f_i(x))$ et $p(y)$ sont les densités de probabilité de $f_i(x)$ et y, et $p(f_i(x), y)$ est la densité de probabilité jointe. Ces densités sont inconnues et doivent être estimées à partir des données. [Torkkola 03], par exemple, utilise des fenêtre de Parsen pour estimer les densités mises en jeu et calculer l'information mutuelle. Une autre approche consiste à discrétiser les données pour se ramener au cas discret. L'information mutuelle s'écrit alors

$$\mathrm{IM}(f_i) = \sum_{a \in \mathcal{X}_i} \sum_{b \in \mathcal{Y}} P(f_i(x) = a, y = b) \log \frac{P(f_i(x) = a, y = b)}{P(f_i(x) = a) P(y = b)}$$

où les probabilités P mises en jeu sont estimées par leur valeurs empiriques [Forman 03].

– Le *cosinus* entre le vecteur $\boldsymbol{x}_i$ des valeurs pour un attribut f_i et le vecteur $\boldsymbol{y}$ des étiquettes des exemple est aussi un indicateur des attributs pertinents : plus la valeur absolue du cosinus est élevée, plus l'attribut est corrélé à l'étiquette. En termes statistiques, si $\boldsymbol{x}_i$ et $\boldsymbol{y}$ sont centrés, ce score correspond à la covariance entre $f_i(x)$ et y normalisée par la déviation de standard de chacune de ces deux variables.

$$|\cos(\boldsymbol{x}_i, \boldsymbol{y})| = \frac{|\mathrm{cov}(f_i(x), y)|}{\sqrt{\mathrm{var}(f_i(x))\,\mathrm{var}(y)}}$$

Le plus souvent, ce score est utilisé pour ne sélectionner qu'un seul attribut. Puis les données sont projetées sur l'espace orthogonal à cet attribut et un nouvel attribut est sélectionné, et ainsi de suite jusqu'à atteindre un critère d'arrêt fixé [Chen 89, Stoppiglia 03]. Avec une telle procédure, la sélection des attributs n'est plus tout à fait univariée.

Tous ces tests ont l'avantage d'être extrêmement rapides à calculer, mais ils ont l'inconvénient de considérer les attributs pris séparément. Ils sont donc incapables d'indiquer qu'un attribut est redondant ou conditionnellement pertinent. Ainsi un attribut redondant sera conservé et un attribut conditionnellement pertinent sera éliminé alors que dans les deux cas on espérerait le comportement inverse.

Pour remédier à cet inconvénient, Relief [Kira 92] prend en compte l'interdépendance des attributs en examinant l'influence d'un attribut et sa corrélation avec la classe de manière locale, au voisinage de chaque exemple. Formellement, associons à tout exemple x son plus proche voisin $\nu(x)$ (resp. $\mu(x)$) de même classe (resp. de classe differente), au sens de la distance définie sur les attributs de $\mathcal{F}$. On définit alors pour tout attribut f_i,

$$\mathrm{Relief}(f_i) = \sum_{x \in \mathcal{E}} |f_i(x) - f_i(\mu(x))| - |f_i(x) - f_i(\nu(x))|$$

Par la suite, [Kononenko 94] a proposé une version étendue de Relief. Cette version nommée ReliefF est en mesure de traiter des bases de données avec plus de deux classes et propose une gestion des valeurs d'attributs manquantes. [Kononenko 94] conseille aussi l'utilisation non pas du plus proche voisin, mais des k plus proches voisins afin d'avoir un score plus robuste. Enfin [Robnik-Šikonja 03] étudie les propriétés des scores Relief et montre ses limites. Tout d'abord, En présence de quelques dizaines (respectivement une centaine) d'attributs non-pertinents, Relief n'est pas en mesure de détecter l'ensemble des attributs pertinents (respectivement le meilleur attribut pertinent). De plus, en présence d'un trop grand nombre d'attributs pertinents, ou en présence d'attributs redondants, le score Relief se retrouve divisé entre tous les attributs pertinents, ce qui conduit à considérer comme non-pertinents les attributs les moins pertinents et les attributs redondants.

7.2.2 Méthodes enveloppe

Les méthodes enveloppe s'attaquent directement au problème combinatoire de la sélection d'attributs (équation 7.2).

Il s'agit de méthodes de type generate-and-test, qui parcourent (un sous-ensemble) du treillis P(F), associant un score à chaque sous-ensemble F considéré dans la liste des candidats, et mettant à jour la liste des candidats en fonction de ces scores.

Recherche gloutonne L'exploration exhaustive de P(F) est naturellement exclue pour des raisons de coût de calcul. De nombreuses heuristiques, essentiellement inspirées de l'optimisation combinatoire sont ainsi utilisées pour restreindre l'ensemble des candidats explorés. L'heuristique la plus simple est la recherche gloutonne : partant du meilleur candidat F_t évalué parmi la liste des candidats à l'étape t, la liste des candidats à l'étape $t+1$ est formée par les sous-ensembles d'attributs les plus « proches » de F_t.

Les approches les plus simples procèdent en profondeur d'abord, soit par ajout, soit par retrait d'un seul attribut au candidat F_t courant. Plus récemment, [Zhang 08a] a proposé une approche alternée combinant ajout et retrait d'attributs (Forward-Backward), dont l'originalité est d'être guidée par une estimation du gain attendu par ajout/retrait d'un attribut.

Marc Boullé utilise aussi une approche Forward-Backward dans un cadre d'inspiration Bayesienne [Boullé 07]. Plutôt que d'ajouter ou retirer le meilleur attribut à chaque étape, il effectue des passes d'ajout et de retrait sur tous les attributs, ordonnés de façon aléatoire ; à chaque fois qu'un attribut améliore le modèle, il est ajouté (ou retiré).

Boullé remarque que moins de cinq passes sont nécessaires pour qu'il n'y ait plus d'améliorations possibles en ajoutant ou retirant un seul exemple, ce qui rend l'approche économe en temps de calcul. En effet, l'évaluation des sous-ensembles d'attributs se fonde sur un classifieur bayesian naïf, dont la mise à jour pour chaque ajout ou retrait d'un attribut coûte $\mathcal{O}(n)$ opérations. Comme le nombre de passes est limité, le coût total de l'algorithme est alors de $\mathcal{O}(dn)$. Ce faible coût autorise à relancer plusieurs fois l'algorithme et à choisir le meilleur sous-ensemble d'attributs rencontré, ou à tous les combiner.

Notons que sous certaines hypothèses concernant les interdépendances des attributs et la granularité de la recherche locale (look-ahead)[2], les méthodes heuristiques déterminent l'optimum global [Margaritis 09].

7.2.3 Méthodes embarquées

Les méthodes embarquées regroupent les approches liant la sélection d'attributs et l'apprentissage d'hypothèse. L'intérêt de ces approches est qu'elles observent la pertinence d'un attribut dans le contexte des autres attributs. Ces approches sont donc en mesure de déterminer les attributs conditionnellement pertinents.

Sélection d'attributs par régularisation Une approche pour embarquer la sélection d'attributs est d'utiliser un terme de régularisation encourageant le choix de solutions *parcimonieuses* (i.e. utilisant peu d'attributs). Le choix le plus général du terme de régularisation (section 2.2) dans le cas particulier d'une hypothese lineaire w sur $\mathbb{R}^d$ est celui de la norme L_2 de w. Mais on voit

[2]Ces hypothèses correspondant en fait à l'inexistence d'optima locaux au sens de l'opérateur de recherche local utilisé.

qu'ajouter ou enlever un attribut de poids w_i très petit ne « coûte » presque rien en terme de minimisation de la norme L_2, alors que le fait de l'enlever contribue à la sélection d'attributs. Formellement, le nombre d'attributs sélectionnés s'exprime en fonction de la « norme » L_0 de w, i.e. de son nombre de composantes non nulles. Le fait de minimiser directement cette quantité pose cependant des problèmes algorithmiques à deux niveaux : ce n'est pas une norme, et son optimisation correspond à un problème combinatoire (celui de la sélection d'attributs en soi). Pour cette raison, la littérature s'est penchée sur le choix de termes de régularisation constituant une bonne approximation de la norme L_0.

Les premiers travaux dans cette direction, dus à Fung, sont fondés sur une régularisation en norme L_1 conduisant à un problème de *programmation linéaire* [Fung 00]. Partant de la formulation standard des SVM, Fung propose d'apprendre l'hypothèse solution du problème

$$\underset{\substack{w\in\mathcal{X},\\ b\in\mathbb{R},\\ \xi\in\mathbb{R}^n}}{\operatorname{argmin}} \|w\|_1 + C\sum_{i=1}^{n}\xi_i \tag{7.3a}$$

$$\text{t.q.} \quad \begin{cases} y_i\left(\langle w, x_i\rangle + b\right) \geq 1-\xi_i, & i=1,\dots,n \\ \xi_i \geq 0 & i=1,\dots,n \end{cases} \tag{7.3b}$$

Tibshirani considère lui aussi une régularisation en norme L_1, mais face à un terme d'erreur de type *somme des carrés* [Tibshirani 94] :

$$\underset{\substack{w\in\mathcal{X},\\ b\in\mathbb{R},\\ \xi\in\mathbb{R}^n}}{\operatorname{argmin}} \|w\|_1 + C\sum_{i=1}^{n}\xi_i^2 \tag{7.4a}$$

$$\text{t.q.} \quad y_i - \left(\langle w, x_i\rangle + b\right) = \xi_i, \qquad i=1,\dots,n \tag{7.4b}$$

Cette formulation, nommée Lasso pour *least absolute shrinkage and selection operator*, vient du fait que Tibshirani s'intéresse plus particulièrment aux problèmes de régression.

Lasso fait référence dans le domaine de la sélection d'attributs, notamment grâce à l'algorithme LARS (Least Angle Regression) [Efron 04] qui permet de calculer tout le chemin de régularisation en un seul passage. Plus précisément, la solution du problème 7.4 dépend de la valeur du paramètre de régularisation C. LARS permet d'apprendre en une seule fois toutes les solutions possibles en un temps de calcul équivalent à la recherche de la solution des moindre carrés, i.e. de la solution de 7.4 avec $C=\infty$.

Des versions hybrides combinant la norme L_1 et la norme L_2 sont proposées par [Chan 07] et permettent d'éviter l'instabilité due à la norme L_1. Enfin, plus récemment, [Langford 09] propose un algorithme capable d'approcher les résultats de Lasso dans un contexte d'apprentissage en ligne.

D'autres critères de régularisation orientés vers la sélection d'attributs ont été proposés : [Bradley 98] utilise par exemple un critère fondé sur une approximation concave de la norme L_0 ; [Weston 03] s'intéresse quand à lui à un critère logarithmique ; ou encore [Zhang 08b] s'attaque plus généralement à des termes de régularisation non convexes. Il faut aussi remarquer que la sélection d'attributs peut bien entendu bénéficier de connaissances *a priori* sur la pertinence des attributs [Helleputte 09].

Les approches à noyaux multiples (MKL pour *Multiple Kernel Learning*) [Lanckriet 04] permettent d'étendre Lasso aux espaces d'hypothèses non-linéaires. Ces approches se placent dans le cadre des SVM (section 2.2) et cherchent la meilleure matrice de noyau K. Cette matrice est choisie

sous la forme

$$K = \sum_{j=1}^{m} \eta_j K_j$$

avec $K_1, \ldots, K_m$ m matrices de noyau, $\eta_1, \ldots, \eta_m$ positifs et $\sum_{j=1}^{m} \eta_j = 1$. Le problème consiste donc à optimiser simultanément les coefficients de lagrange $\alpha_1, \ldots, \alpha_n$ et les coefficients $\eta_1, \ldots, \eta_m$. Cette formulation initiale des MKL a été modifiée à plusieurs reprises [Bach 04, Sonnenburg 06, Zien 07, Rakotomamonjy 08] de façon à obtenir un problème d'optimisation plus simple.

Le cadre des MKL inclut le cadre de la sélection d'attributs. En effet, les MKL ont pour objectif, de façon générale, la sélection du bon noyau. Mais si chaque noyau correspond à l'utilisation d'un sous-ensemble d'attributs, alors les MKL permettent la sélection des sous-ensemble d'attributs pertinents.

Les MKL intéressent toujours la communauté de l'apprentissage. Dans les avancés récentes, on retrouve notamment [Bach 08], qui étend les MKL au cas où les noyaux sont organisés de façon hiérarchique, et [Varma 09], qui propose une formulation plus générale des MKL. La formulation de [Varma 09] autorise par exemple l'utilisation d'un noyau K défini comme un produit des noyaux $K_1, \ldots, K_m$. Mais cette formulation prend alors le risque de devoir résoudre des problèmes d'optimisation non-convexes.

Apprendre un modèle pour sélectionner des attributs Les modèles appris par les approches mixant apprentissage de modèle et sélection d'attributs permettent d'estimer la pertinence des attributs. Ainsi, il est naturel de considérer que seuls les attributs intervenant dans un arbre de décision sont pertinents ; un autre algorithme d'apprentissage (Machine à vecteurs support ou réseau neuronal) peut ensuite être utilisé à partir de ce sous-ensemble. Notons que ceci correspond à utiliser le score de l'arbre de décision (critère d'entropie) comme une méthode de type filtre. L'intérêt par rapport aux méthodes filtre traditionnnelles est que le criteère est calculé conditionnellement aux attributs déjà sélectionnés, comme dans une méthode de style enveloppe.

Dans le même esprit, l'approche proposée par Alphonse et Stroppa [Alphonse 02] résout de petits problèmes d'apprentissage pour filtrer les littéraux d'un problème d'apprentissage en logique inductive (ILP). Concrètement, à chaque exemple x est associé un problème à instances multiples $mip(x)$ tel que sélectionner les attributs pertinents de $mip(x)$ revienne à sélectionner les litéraux informatifs de x. Une hypothèse $h_{mip(x)}$ résolvant le problème $mip(x)$ est alors apprise à l'aide d'un algorithme glouton ou d'un algorithme d'espérance-maximisation. Les attributs pertinents sont ceux utilisés par $h_{mip(x)}$. Ainsi, l'apprentissage glouton ou par espérance-maximisation permet à la fois de sélectionner les attributs tout en apprenant une hypothèse.

Des algorithmes d'apprentissage plus sophistiqués sont aussi utilisés pour mettre en évidence les attributs pertinents. Par exemple, le résultat d'une forêt aléatoire peut être utilisé comme une méthode filtre [Breiman 84, Rogers 05], ordonnant les attributs en fonction de leur score gini moyen sur les nœuds de la forêt. De même les poids d'une combinaison linéaire de plusieurs noyaux fournissent un ordonnancement des attributs [Xu 09].

Apprendre un modèle pour parcourir le treillis des sous-ensembles d'attributs L'information fournie sur les attributs lors de l'apprentissage peut être utilisée de façon plus subtile encore. Ainsi, [Tuv 09] alterne des phases d'apprentissage à partir de forêts aléatoires qui servent à détecter les attributs pertinents, avec des phases d'élimination des attributs redondants.

Dans d'autres cas, le score obtenu est utilisé pour se promener dans l'ensemble des parties de $\mathcal{F}$ à la façon des méthodes enveloppe. C'est le cas notamment de SVM-RFE (pour SVM Recursive Feature Elimination) [Guyon 02] qui utilise comme score le poids des attributs d'une *machine à vecteurs de support* (SVM pour *Support Vector Machine*) linéaire, ou encore de [Shen 08] qui se fonde sur la modification de comportement d'un SVM lorsqu'un attribut est perturbé. Ces deux approches procèdent par élimination récursive des attributs. En partant du sous-ensemble d'attributs F contenant tous les attributs, ces approches alternent deux phases :

- apprendre un SVM utilisant uniquement les attributs contenus dans F ;
- retirer de F les attributs les moins intéressants d'après le score fondé sur le SVM appris.

7.3 Discussion

La sélection d'attributs est un problème difficile principalement pour deux raisons :

- le problème d'optimisation associé est un problème combinatoire qui n'est pas sous-modulaire et ne peut donc pas être simplifié ;
- la fonction à minimiser, l'erreur en généralisation, est inconnue. Cette fonction ne peut qu'être estimée.

Les différentes méthodes qui se confrontent au problème de la sélection d'attributs se limitent donc soit à résoudre un problème plus simple mais dont l'optimum est proche du problème initial, soit à ne tester qu'un nombre restreint de sous-ensembles d'attributs choisis selon une heuristique.

Notamment, les méthodes filtre et embarquées considèrent chaque attribut dans un contexte de type tout ou rien : l'attribut est considéré soit de manière isolée (et elles ne détectent pas les interactions entre attributs), soit dans le contexte de tous les autres attributs (et elles sont perturbées par les attributs redondants ou non-pertinents). Seules les méthodes enveloppe, qui considèrent les sous-ensembles d'attributs, ont une chance de voir les attributs sous leur meilleur angle. Mais le prix à payer est qu'il leur faut considérer les bons sous-ensembles d'attributs, c'est à dire explorer d'une manière ou d'une autre le treillis P(F).

Les approches enveloppe sont quant à elles face au classique dilemme *Exploration/Exploitation* :

- *L'exploration* requiert de pouvoir tester des sous-ensembles d'attributs nouveaux, loin des sous-ensembles qui constituent les meilleurs candidats locaux, dans le but d'échapper à la convergence prématurée vers des optima locaux de mauvaise qualité ;
- *L'exploitation* consiste à visiter le voisinage des meilleurs candidats courants de façon à déterminer l'optimum local correspondant.

Le prochain chapitre présente divers approches conçues pour se confronter au dilemme Exploration/Exploitation.

Chapitre 8

Exploration vs Exploitation et Bandits Manchots

La question abordée dans ce chapitre est celle du dilemme entre Exploration et Exploitation, que nous rencontrons dans le cadre de l'optimisation combinatoire et de la sélection d'attributs. Le cadre d'analyse proposé est celui des algorithmes de bandits manchots, bien étudiés en statistique [Robbins 52, Berry 85], et qui intervient de manière essentielle dans plusieurs problématiques d'apprentissage tels l'apprentissage en ligne [Littlestone 89] ou l'apprentissage par renforcement [Sutton 98]. Nous présenterons la position du problème avant de décrire et de discuter les deux familles d'algorithmes de la littérature, consacrées respectivement aux problèmes de choix d'options (section 8.2), et de choix de séquences d'option (section 8.4).

8.1 Le problème du bandit manchot

Cette section présente le cadre du problème du bandit manchot.

Le probleme du bandit manchot [Robbins 52, Lai 85, Auer 02] fait intervenir k bras indépendants. Chaque bras i est caractérisé par une loi de probabilité ν_i sur $[0, b]$ d'espérance μ_i, déterminant la récompense obtenue lorsque l'on joue ce bras. Ces lois sont inconnues du joueur, et fixes dans le temps. On note μ^* l'espérance du meilleur bras i^*.

$$i^* = \underset{i=1,\ldots,k}{\operatorname{argmax}} \mu_i$$

$$\mu^* = \mu_{i^*} = \max_{i=1,\ldots,k} \mu_i$$

A chaque pas de temps t, le joueur choisit un bras i_t et recoit une récompense r_t. Son but est naturellement de maximiser la récompense cumulée

$$\sum_{t=1}^{T} r_t$$

ou de manière équivalente à minimiser son regret défini par sa perte par rapport au gain de l'oracle (qui aurait toujours choisi le bras i^* de gain maximal). L'espérance de ce regret, qui doit être minisée est ainsi donnée par

$$\mathbb{E}[R_T] = \sum_{t=1}^{T} (\mu_{i^*} - \mu_{i_t})$$

Le joueur, qui ne connait pas les loies ν_i *a priori*, mais peut les estimer au cours du temps, est ainsi confronté à un dilemme Exploration/Exploitation :

- *Exploitation* : une façon simple de diminuer le regret est de ne pas prendre de risque et de jouer le bras qui a conduit à la meilleure récompense en moyenne lors des itérations précédentes, notée $\hat{\mu}_i$.
  Cette stratégie entraine cependant un regret qui croît linéairement avec le temps en $\mathcal{O}(T \times (\mu^* - \mu_i)))$, si le meilleur bras empirique n'est pas le bon ($i \neq i^*$) ;
- *Exploration* : le joueur doit ainsi consacrer une certaine partie de ses essais à découvrir s'il existe un bras meilleur que celui qu'il a identifié jusqu'ici. L'exploration est evidemment coûteuse.

De nombreuses variantes du cadre du bandit manchot (MAB pour Multi-Armed Bandit) concernent l'horizon temporel (fini ou non), le nombre d'options (grand ou petit en fonction de l'horizon temporel), et la dépendance entre les bras. Le lecteur est réferré à la thèse de S. Bubeck [Bubeck 10a] pour une présentation générale du problème de bandit manchot, qui sort du cadre de la presente thèse.

Dans le contexte qui a été considéré (bras et tirages indépendants, gains fixes dans le temps), les résultats théoriques montrent que le regret optimal est en $\mathcal{O}(\log T)$ [Lai 85].

8.2 La famille des algorithmes Upper Confidence Bound

Un algorithme atteignant la borne théorique a été proposé par Auer et al [Auer 02], appelé *Upper Confidence Bound* (UCB). Le principe de cet algorithme est décrit de manière intuitive par "l'optimisme en face de l'inconnu", conduisant à privilégier le bras tel que son gain *peut*, sur la base des informations disponibles, etre celui du meilleur bras.

Formellement, à tout bras i est associé un intervalle de confiance $(\hat{\mu}_i - \epsilon_i, \hat{\mu}_i + \epsilon_i)$, où la confiance ϵ_i dépend du nombre de fois n_i que le bras i a été joué. L'algorithme UCB consiste ainsi à sélectionner le bras i maximisant la borne supérieure de l'intervalle ci-dessus, soit : $\hat{\mu}_i + \epsilon_i$, avec $\epsilon_i \propto \sqrt{\frac{\log t}{n_i}}$.

L'algorithme UCB1 proposé par Auer et al. [Auer 02] est donné par la sélection à l'itération t de l'option maximisant :

$$\hat{\mu}_i + \sqrt{\frac{2b^2 \log t}{n_i}} \tag{8.1}$$

où $\hat{\mu}_i$ et n_i correspondent respectivement au gain empirique et au nombre de fois que le i-ème bras a été joué. Cet algorithme garantit un regret logarithmique en t, dépendant inversement de la marge (écart entre le meilleur et le second meilleur gain) et variant linéairement avec le nombre de bras ; ce résultat découle de la borne de Hœffding.

$$\mathbb{E}[R_T] \leq 8 \sum_{i:\mu_i < \mu^*} \left(\frac{b^2}{\mu^* - \mu_i} \right) \log(T) + \left(1 + \frac{\pi^2}{3} \right) \sum_{i=1}^{k} (\mu^* - \mu_i) \tag{8.2}$$

8.2.1 Utilisation de la variance

Un raffinement de l'algorithme UCB1 consiste à tenir compte de la variance des gains d'un bras : plus la variance d'un bras est faible, moins il faut d'itérations pour se rendre compte qu'il n'est pas optimal. Partant de cette remarque, [Auer 02] et [Audibert 09] ont proposé des versions d'UCB qui prennent en compte la variance empirique $\hat{\sigma}^2$ des différents bras.

Dans UCB1, b^2 apparaît dans le terme d'exploration en temps que borne supérieure de la variance (à un facteur près). Auer et al. [Auer 02] proposent de remplacer b^2 par une estimation empirique de cette borne. L'algorithme ainsi définit (UCB-tuned) sélectionne le bras maximisant

$$\hat{\mu}_i + \sqrt{\frac{c_e \log t}{n_i} \min\left(\frac{b^2}{4}, \hat{\sigma}_i^2 + \sqrt{\frac{2c_e b^2 \log t}{n_i}}\right)} \tag{8.3}$$

Notons que c_e est une constante qui n'apparaît pas dans la version originale de l'article, mais qui est utilisée pour contrôler le niveau d'exploration. Cet algorithme est très efficace en pratique lorsque c_e est convenablement ajustée. En revanche, il n'existe pas de résultat théorique sur le regret associé.

Pour introduire un estimateur de la variance, l'algorithme UCB-V part de la borne de Bernstein, au lieu de la borne de Hœffding. Cette algorithme, proposé par Audibert et al. [Audibert 09], sélectionne le bras maximisant

$$\hat{\mu}_i + \sqrt{\frac{2c_e \log t}{n_i}\hat{\sigma}_i^2} + c\frac{3c_e b \log t}{n_i} \tag{8.4}$$

Sous certaines conditions sur c et c_e ($c = 1$ et $c_e > 1$), UCB-V permet d'avoir un regret vérifiant

$$\mathbb{E}[R_T] \leq C \sum_{i:\mu_i<\mu^*} \left(\frac{\sigma_i^2}{\mu^* - \mu_i} + 2b\right) \log(T) \tag{8.5}$$

avec C une constante dépendant de c_e. La différence avec le regret 8.2, est que les termes en $\frac{1}{\mu^*-\mu_i}$ sont pondérés par la variance σ_i de chaque bras au lieu de sa borne supérieure. UCB-V a donc un regret meilleur que UCB1, lorsque certains bras non-optimaux ont une petite variance.

De plus, [Audibert 09] démontre que, quel que soit c, si $c_e < 1$ (ou quel que soit c_e, si $c \times c_e < 1/3$), alors il existe un ensemble de distributions $\nu_1, \ldots, \nu_k$ sur les récompenses, dépendant de T, tel que UCB-V souffre un regret polynomial. Il faut donc faire attention dans le choix des paramètres c et c_e.

8.2.2 Many-armed Bandits

Par construction, les algorithmes UCB doivent commencer par essayer au moins une fois chaque bras. Dans le cas où le nombre de bras est grand (ou infini) devant le nombre d'itérations, les algorithmes UCB sont ainsi biaisés vers l'exploration.

Plusieurs algorithmes ont été proposés pour remédier à ce biais. Dans le cas particulier où les lois des bras sont des lois de Bernoulli, [Teytaud 07], propose deux algorithmes. Le premier algorithme est une variante d'UCB regroupant les bras sous forme arborescente ; le deuxième algorithme consiste principalement, quant à lui, à jouer le bras courant tant qu'il retourne 1 comme récompense. [Teytaud 07] remarque que suivant la difficulté du problème considéré, l'un ou l'autre de ces deux algorithmes est le plus adapté.

Une heuristique a été proposée simultanément par [Coulom 07] et [Chaslot 07]. Cette heuristique connue sous le nom d'élargissement progressif (EP) ou de *progressive unpruning* est étudiée théoriquement dans [Wang 08] et est appliquée par [Rolet 09] dans le cadre de l'apprentissage actif.

L'idée consiste à augmenter graduellement l'ensemble des bras considérés au fur et à mesure du temps. Formellement, le nombre $k(t)$ de bras considéré varie comme $\log t$ ou t^a, avec $a < 1$ paramètre de l'algorithme. Le choix des bras considérés est uniforme ou fait intervenir des connaissances du domaine (section 9.2.1).

8.3 Algorithmes Monte-Carlo Tree Search

Le problème du bandit manchot a été généralisé au choix d'une séquence d'options (ou du chemin dans un arbre) par Kocsis et Szepesvári [Kocsis 06]. Ce cadre est celui dit du Monte-Carlo Tree Search : étant donné un arbre, dont les feuilles portent des récompenses données par des variables aléatoires de lois de probabilité ν_i, l'objectif est de découvrir la feuille de récompense maximale, en parcourant un certain nombre de chemins dans cet arbre. Par rapport au contexte du bandit manchot, le choix d'un bras est remplacé par le choix d'un chemin. Présentons tout d'abord le cadre général des algorithmes Monte-Carlo Tree Search (MCTS) avant d'introduire l'extension d'UCB aux arbres, appelée UCT [Kocsis 06].

8.3.1 Recherche dans un arbre

Soit $\mathcal{B}$ un arbre de racine F_0. Notons $\mathcal{B}_1, \ldots, \mathcal{B}_k$ les sous-arbres de $\mathcal{B}$ dont la racine est un fils de F_0 ($\mathcal{B} = \text{arbre}(F_0, \{\mathcal{B}_1, \ldots, \mathcal{B}_k\})$). L'objectif premier des MCTS est de déterminer lequel de ces sous-arbres contient la feuille optimale sachant que l'on n'a pas assez de temps pour tester toutes les feuilles. Une solution basique consiste à tester des branches au hasard pour chaque sous-arbre $\mathcal{B}_i$ et de sélectionner le sous-arbre qui a donné les meilleurs résultats. Le principe des MCTS [Chaslot 06, Coulom 06, Kocsis 06] est de biaiser les branches testées dans chaque sous-arbre de façon à trouver plus rapidement le sous-arbre contenant la feuille optimale.

L'intuition consiste à créer un mécanisme hiérarchique d'Exploration/Exploitation. Les premiers chemins parcourus (appelés itérations ou simulations) fournissent une information sur différents nœuds outre la feuille évaluée. Ces informations vont servir à guider le choix des chemins suivants, en respectant un équilibre entre exploitation (biais vers les sous-arbres donnant les meilleurs résultats) et exploration. L'algorithme résultant suit ainsi le schéma suivant :

1. choix d'un chemin : sélection d'un nœud ou d'une feuille réalisant un compromis Exploration/Exploitation (en partie d'après les résultats des itérations précédentes et en partie aléatoirement) ;
2. évaluation de la feuille ;
3. mise à jour des informations sauvegardées.

8.3.2 Sélection d'une feuille

Pour sélectionner une feuille lors d'une itération, l'algorithme part de la racine de l'arbre et passe de nœud fils en nœud fils jusqu'à aboutir à une feuille. Pour passer d'un nœud F donné à son fils, l'algorithme doit choisir un fils qui soit un bon compromis entre exploration et exploitation comme dans le cadre des MAB (Sec 8.1). Ce compromis se fonde sur une valeur associée à chaque

fils s de F en fonction des informations sauvegardées (typiquement la récompense moyenne obtenue pour les itérations précédentes passant par s). Les divers MCTS diffèrent à la fois sur la gestion de ce compromis entre exploration et exploitation, mais aussi sur la manière d'associer une valeur à un nœud.

8.3.3 Informations sauvegardées

L'idéal est de conserver tous les chemins ayant conduit à une feuille ainsi que la récompense associée. Mais le plus souvent ces informations sont trop volumineuses, ce qui conduit des algorithmes tels que [Coulom 06] à conserver les informations uniquement pour des nœuds proches de la racine. Cet ensemble de nœud est couramment appelé *sous-arbre de recherche* et ne grandit au maximum que d'un nœud par itération.

8.3.4 Extensions

Les MCTS ont été appliqués de manière très générale et leur étude approfondie sort du cadre de cette thèse. Sans exhaustivité nous mentionnerons ainsi quelques variantes de MCTS :

- la récompense peut être fournie tout au long du chemin (une récompense partielle par nœud), la récompense globale étant une somme pondérée des récompenses partielles [Kocsis 06, Bubeck 10b] ;
- la récompense en une feuille peut être non déterministe [Bubeck 10b] ;
- l'arbre peut faire intervenir une typologie des nœuds, par exemple dans le cas d'un jeu à deux joueurs, où les critères de choix du nœud fils depend du nœud courant [Gelly 07].

Une dernière remarque porte sur l'information retournée par l'algorithme. De façon standard, les MCTS sont présentés comme un moyen de choisir le meilleur sous-arbre $\mathcal{B}_i$ (i.e. le meilleur fils de la racine). Cela provient du fait qu'ils sont principalement utilisés dans le cadre des jeux et que donc après avoir joué un coup il est pertinent d'attendre de connaître la réponse jouée par l'adversaire avant d'annoncer le coup suivant. Mais en pratique les MCTS construisent petit à petit toute la trajectoire vers la meilleure feuille, et peuvent donc être utilisés pour déterminer entièrement cette trajectoire (et pas seulement le premier coup).

8.4 UCB applied to Trees

L'apport du contexte de bandit manchot dans le cadre des MCTS, du à Kocsis et Szepesvári, est de résoudre le compromis Exploration/Exploitation en chaque nœud de l'arbre, utilisant la formule UCB1 (équation 8.1) [Kocsis 06]. Cette approche requiert qu'une récompense empirique soit associée à chaque nœud, en prenant la moyenne des récompenses reçues pour les feuilles correspondantes.

L'analyse de l'algorithme montre sa convergence asymptotique (la probabilité de choisir le bon nœud fils à la racine tend vers 1 lorsque le nombre d'itérations tend vers l'infini [Kocsis 06]) mais nous ne disposons pas de borne sur le regret à notre connaissance.

Concrètement, l'algorithme UCT (pour *UCb applied to Trees*) fait intervenir trois aspects principaux : le calcul de l'information associée à un nœud ; le développement graduel de l'arbre au cours des itérations ; l'algorithme de choix entre les fils d'un nœud.

8.4.1 Évaluation d'un nœud

Une solution pour choisir le bon nœud fils pour chaque nœud serait de connaître le score V^* défini récursivement sur les nœuds par

$$V^*(F) = \begin{cases} \text{récompense}(F) & \text{si } F \text{ est une feuille} \\ \max\limits_{F' \in \text{fils}(F)} V^*(F') & \text{sinon} \end{cases} \tag{8.6}$$

Le meilleur nœud fils F^* est alors celui maximisant V^* :

$$F^* = \operatorname*{argmax}_{F' \in \text{fils}(F)} V^*(F') \tag{8.7}$$

UCT converge justement parce que la valeur associée à un nœud ($\hat{\mu}$ ou le score UCB associé) converge vers V^*. Pourtant cette valeur est définie comme une moyenne sur les itérations et non pas comme un maximum. En fait, cette moyenne converge vers un maximum car au fil des itérations UCT choisit de plus en plus souvent la meilleure branche ; la moyenne sur les itérations tend donc vers la moyenne sur la meilleure branche, i.e. vers le maximum.

Le fait de fonder les choix sur la moyenne et non pas le maximum n'empêche pas la convergence mais a tout de même un inconvénient : UCT a tendance à beaucoup explorer. Comme la valeur d'un nœud est une moyenne, elle est sous-estimée et cela augmente donc les chances de considérer comme meilleur un nœud qui ne l'est pas et par suite de le sélectionner.

8.4.2 Choix d'un nœud dans l'arbre

Ce choix contrôle le compromis entre exploration et exploitation. L'usage de l'équation UCB (équation 8.3) entraine un excès d'exploration, amplifié par le fait qu'elle intervient à tous les niveaux de l'arbre. La littérature [Gelly 06] et [Teytaud 09] utilise ainsi de préférence UCB1-tuned (équation 8.3) et UCB-V (équation 8.4). Ces deux variantes d'UCB sont choisies car elles utilisent la variance empirique. Ainsi l'algorithme sélectionne moins souvent les nœuds-fils ayant une faible variance et consacre plus de temps à estimer la valeur des nœuds avec une variance forte.

8.4.3 Développement graduel de l'arbre

UCT, tel que présenté par [Kocsis 06], conserve les informations pour tous les nœuds rencontrés lors d'une itération. Comme pour les MCTS en général, ces informations peuvent être trop volumineuses. Nous nous concentrerons ainsi dans la suite sur la variante d'UCT développée par [Gelly 06], qui fait grandir graduellement l'arbre de recherche, en partant d'un arbre limité au nœud racine, et en ajoutant un seul nœud à chaque itération. Ce faisant, UCT fait grandir un sous-arbre de recherche asymétrique (figure 8.1).

Chaque itération débute à la racine et va de nœud fils en nœud fils tant que faire se peut. Tant que le nœud F appartient au sous-arbre de recherche, le nœud fils sélectionné est le meilleur selon UCB-V. Si F est une feuille de l'arbre de recherche, si ce nœud peut être évalué il est évalué et l'évaluation de tous les nœuds du chemin est mise à jour ; sinon, un nœud fils de F est sélectionné et ajouté à l'arbre de recherche ; ce nœud est complété aléatoirement de façon à pouvoir être évalué, et cette évaluation sert de même à mettre à jour l'évaluation de tous les nœuds du chemin.

Pour chaque nœud F du chemin, les informations mises à jour sont la moyenne empirique $\hat{\mu}_F$, la variance empirique $\hat{\sigma}_F$ et le nombre T_F d'itérations passées par F.

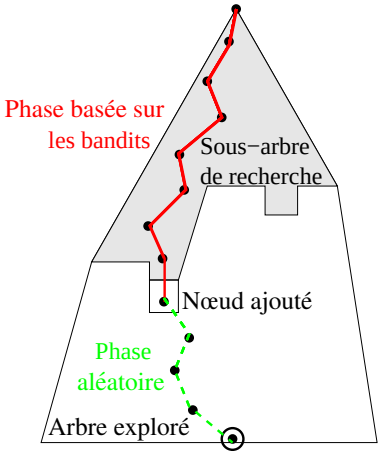


FIG. 8.1 – Une itération d'UCT : dans le sous-arbre de recherche, le chemin est choisi en fonction du critère UCB ; le premier nœud rencontré n'appartenant pas au sous-arbre de recherche est ajouté au sous-arbre de recherche ; en dehors du sous-arbre de recherche le chemin est choisi aléatoirement.

8.4.4 Applications

UCT a été appliqué aussi bien dans le cadre des jeux (Go [Gelly 07], Havannah [Teytaud 09]), de l'apprentissage actif [Rolet 09] ou du découpage de matrices lors d'une multiplication [de Mesmay 09]. Pour toutes ces applications, UCT est utilisé pour rechercher la meilleure feuille (le meilleur nœud terminal) dans un graphe orienté et non pas un arbre. Mais ce sont des graphes orientés acycliques, qui peuvent donc se réécrire comme des arbres à condition d'en dupliquer certains nœuds. Le fait de les traiter comme des graphes permet de factoriser l'information en ne procédant pas à la duplication.

Concrètement, l'enjeu concerne la mise à jour de la valeur des nœuds : la valeur calculée dans une feuille peut être utilisée pour mettre à jour la valeur de tous les nœuds de tous les chemins menant effectivement à cette feuille, ce qui est le cas par [de Mesmay 09, Rolet 09], et non par [Gelly 07] et dans la suite du manuscrit. L'avantage d'une mise à jour plus fréquente est bien sûr d'accélérer la convergence. En contrepartie, i.e. si le facteur de branchement du graphe est élevé, la mise à jour peut faire intervenir un nombre exponentiel de chemins.

Un second aspect concerne l'usage d'informations externes ou issues de la recherche [Gelly 07, Teytaud 09] afin de guider la recherche. Un exemple d'informations externes est l'usage de *motifs* (patterns) par les auteurs du jeu de Go, ou le choix des coups lors de la phase Monte Carlo. L'usage d'information acquise pendant la recherche comprend en particulier la construction des vecteurs RAVE (Rapide Action Value Estimation) sur lequel nous reviendrons au chapitre suivant.

8.5 Résumé

L'optimisation combinatoire associée au problème de la sélection d'attributs soulève le dilemne entre exploration et exploitation. L'étude de ce dilemne se fait au travers du cadre des bandits manchots, où un joueur fait face à une machine à sous à k bras, dont les récompenses sont distribuées selon $\nu_1, \ldots, \nu_k$ d'espérances respectives $\mu_1, \ldots, \mu_k$. Le joueur ne connaît pas les distributions ou

les espérances des bras, mais il veut maximiser son gain, i.e. jouer le plus souvent possible le bras i^* d'espérance maximale.

$$\mu_{i^*} = \mu^* = \max_{i=1,\dots,k} \mu_i$$

Le problème des bandits manchots correspond parfaitement au dilemne entre exploration et exploitation, puisque le joueur doit choisir entre utiliser le bras qui a fourni les meilleures récompenses dans le passé (exploiter), ou tester un autre bras (explorer). De plus, lorsque l'objectif du joueur est de maximiser la somme des récompenses obtenues au cours de T tirages, les algorithmes de la famille UCB (pour *Upper Confidence Bound*) fournissent des résultats optimaux à une constante près.

Mais lorsque les bras à considérer correspondent aux feuilles d'un arbre, plutôt que de mettre cet arbre à plat et d'utiliser un algorithme UCB, il est possible de tenir compte de la structure des bras. Ainsi l'algorithme UCT (pour *UCb applied to Trees*) se promène dans l'arbre considéré en mettant un UCB dans chaque nœud de l'arbre. L'avantage d'une telle approche est que l'algorithme converge plus rapidement lorsque la structure apporte de l'information, i.e. lorsque les meilleurs bras sont regroupés dans un même sous-arbre.

Or, dans le cadre de la sélection d'attributs, les sous-ensembles d'attributs peuvent justement se regrouper dans un treillis. Le chapitre suivant s'intéresse donc à l'utilisation d'UCT pour rechercher dans ce treillis les sous-ensembles d'attributs minimisant l'erreur en généralisation.

Chapitre 9

De la sélection d'attributs comme un problème d'apprentissage par renforcement

Ce chapitre présente notre approche pour la sélection d'attributs : FUSE (pour *Feature Uct SElection*). L'objectif de FUSE est de s'attaquer à la sélection d'attributs dans les cas les plus difficiles : présence d'attributs redondants, présence de nombreux attributs non pertinents et présence d'attributs conditionnellement pertinents. FUSE est donc une approche de type enveloppe.

Le point clef est de proposer un bon compromis entre exploration et exploitation. Pour ce faire, FUSE considère la sélection d'attributs comme un problème d'apprentissage par renforcement sur un processus de décision markovien (section 9.1) et adapte UCT pour résoudre ce problème d'apprentissage par renforcement (section 9.2).

9.1 Formalisation en terme de processus de décision markovien

Cette partie s'intéresse à la formalisation de la sélection d'attributs en terme de processus de décision markovien (MDP). L'objectif devient alors de trouver une politique optimale pour ce MDP. Fréquemment dans la littérature, le problème de la sélection d'attributs (équation 7.2) est simplifié pour se ramener à un problème continu, voire convexe. Au contraire, la vision proposée dans cette section permet d'attaquer directement le problème combinatoire de la sélection d'attributs.

Définition du processus de décision markovien À un problème de sélection d'attributs sur un ensemble d'attributs $\mathcal{F}$ est associé le MDP $\mathcal{M}$ défini comme suit (figure 9.1). On note $\bar{\mathcal{F}}$ l'ensemble d'attributs additionné d'un attribut *stop* noté f_s. Les composantes de $\mathcal{M}$ sont alors :

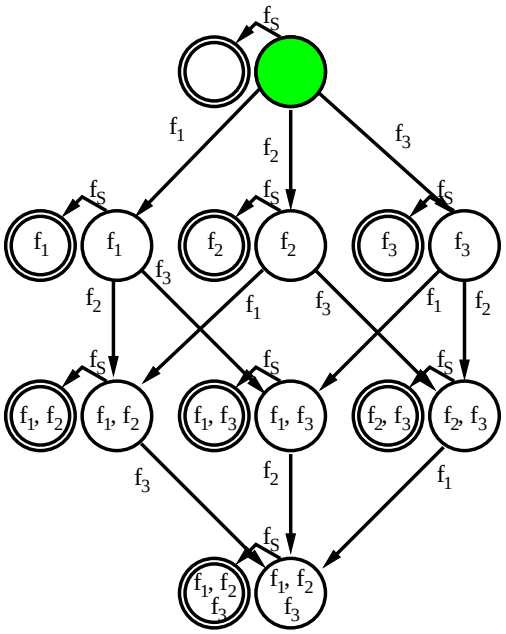


FIG. 9.1 – MDP $\mathcal{M}$ correspondant à un ensemble de trois attributs $\mathcal{F} = \{f_1, f_2, f_3\}$. L'état en vert est l'état initial. Tout état final peut être atteint depuis l'état initial, donc tout sous-ensemble d'attributs peut être obtenu en parcourant ce MDP.

états : $\mathcal{S} = 2^{\bar{\mathcal{F}}}$

états initiaux : $\mathcal{S}_i = \{\varnothing\}$

états finaux : $\mathcal{S}_f = \{F \in \mathcal{S} \text{ t.q. } f_s \in F\}$

actions : $A = \{\text{ajoute}(f), f \in \bar{\mathcal{F}}\}$

fonction de transition : p :
$$\begin{array}{rcl} \mathcal{S} \times A \times \mathcal{S} & \rightarrow & \mathbb{R}^+ \\ (F, \text{ajoute}(f), F') & \mapsto & \begin{cases} 1 & \text{si } F' = F \cup \{f\} \text{ et } f_s \notin F \\ 0 & \text{sinon} \end{cases} \end{array}$$

récompense : r :
$$\begin{array}{rcl} \mathcal{S}_f & \rightarrow & \mathbb{R} \\ F & \mapsto & \mathbf{Err}\left(\mathcal{A}\left(F, \mathcal{E}\right)\right) \end{array}$$

Cette définition de $\mathcal{M}$ conduit à plusieurs remarques. Tout d'abord, on ne considère que les actions qui consistent à ajouter un attribut. Mais cela ne pose pas de problème en théorie, puisque tout sous-ensemble d'attributs (et donc en particulier le sous-ensemble d'attributs optimal) peut se construire en ajoutant successivement chaque attribut qu'il contient.

Ensuite, $\mathcal{M}$ est déterministe : la probabilité de transition d'un état F vers un état F' en appliquant l'action ajoute(f) est non nulle uniquement pour l'état $F' = F \cup \{f\}$.

Enfin, seuls les états terminaux ont une récompense et l'objectif est de la minimiser. En principe cette récompense est l'erreur en généralisation du sous-ensemble d'attributs correspondant à l'état courant ; en pratique, il s'agira naturellement d'une estimation de cette erreur en généralisation[1].

[1] L'attribut f_s n'est pas utilisé au cours de l'apprentissage. L'erreur en généralisation du sous-ensemble d'attributs F notée $\mathbf{Err}\left(\mathcal{A}\left(F, \mathcal{E}\right)\right)$ est donc en réalité l'erreur en généralisation du même sous-ensemble d'attributs auquel on a retiré f_s : $\mathbf{Err}\left(\mathcal{A}\left(F \setminus \{f_s\}, \mathcal{E}\right)\right)$

Politique optimale Résoudre l'équation 7.2 est alors équivalent à trouver un chemin menant de l'état initial de $\mathcal{M}$ à l'état final associé au sous-ensemble d'attributs optimal. Ce chemin peut se définir à partir d'une *politique*, c'est à dire d'une fonction $\pi : \mathcal{S} \rightarrow A$ indiquant dans chaque état l'action à choisir. Notons F_π l'état final atteint en partant de l'état initial et en suivant la politique π. L'objectif de la sélection d'attributs est donc de trouver la politique optimale $\pi^\star$ qui minimise $\mathbf{Err}\left(\mathcal{A}\left(F_\pi, \mathcal{E}\right)\right)$:

$$\pi^\star = \underset{\pi}{\operatorname{argmin}} \ \mathbf{Err}\left(\mathcal{A}\left(F_\pi, \mathcal{E}\right)\right) \tag{9.1}$$

D'après le principe d'optimalité de Bellman [Bellman 57], $\pi^\star$ est donnée par :

$$\pi^\star(F) = \underset{f \in \bar{\mathcal{F}} \setminus F}{\operatorname{argmin}} \ V^\star(F \cup \{f\}) \tag{9.2}$$

où $V^\star$ est la fonction de valeur optimale définie par :

$$V^\star(F) = \begin{cases} \mathbf{Err}\left(\mathcal{A}\left(F, \mathcal{E}\right)\right) & \text{si } F \text{ est final} \\ \min\limits_{f \in \bar{\mathcal{F}} \setminus F} V^\star(F \cup \{f\}) & \text{sinon} \end{cases} \tag{9.3}$$

Les équations de Bellman indiquent comment calculer la politique optimale : $V^\star$ est calculé de façon récursive en partant des états finaux (équation 9.3), puis la politique optimale est déduite de l'équation 9.2. Mais le MDP $\mathcal{M}$ comporte un nombre exponentiel d'états ($2^{|\bar{\mathcal{F}}|}$), il est donc impossible d'éffectuer le calcul complet lorsque l'on est confronté à des bases de données possédant plusieurs centaines d'attributs. La solution proposée est d'approcher graduellement $\pi^\star$ grâce à l'algorithme UCT décrit dans la section 8.4.

UCT ne définit une politique proche de la politique optimale que pour les quelques états qu'il a visité suffisamment souvent. En dehors de ces états, UCT propose une politique de mauvaise qualité, puisque cette politique est fondée sur quelques simulations uniquement, voire sur aucune simulations. Mais l'objectif de la sélection d'attributs est de trouver le bon état final ou un chemin y menant. Donc il n'est pas nécessaire de connaître la politique optimale pour tous les états, il suffit de connaître la politique optimale pour les états sur un chemin menant à l'état optimal ; et c'est exactement ce qu'UCT va rechercher.

9.2 FUSE : Approximation par recherche Monte-Carlo dans un arbre

L'algorithme FUSE (pour *Feature Uct SElection*) proposé approche la politique optimale du MDP $\mathcal{M}$ en appliquant l'algorithme UCT. Cette section décrit les adaptations d'UCT développées dans ce but.

Avant de détailler ces adaptations, il faut remarquer qu'UCT est conçu pour trouver une branche optimale dans un arbre. Or $\mathcal{M}$ n'est pas un arbre. Mais en pratique UCT s'applique parfaitement au cas de graphes tels que ceux que l'on croise dans le cadre des jeux (Go [Gelly 07], Havannah [Teytaud 09]) ou de l'apprentissage actif [Rolet 09] (section 8.4).

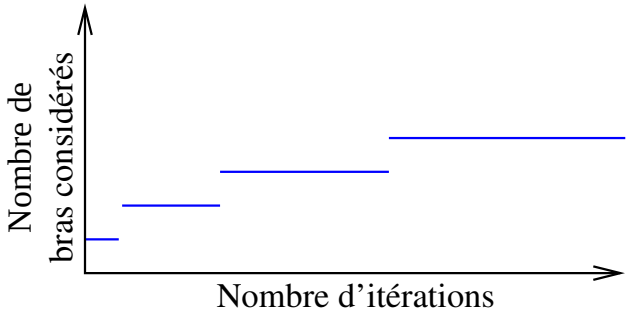


FIG. 9.2 – Nombre de bras considérés par UCB en fonction du nombre d'itérations. Le nombre de bras est restreint par l'heuristique d'*élargissement progressive*.

9.2.1 Politique dans l'arbre de recherche : prise en compte du facteur de branchement élevé

Une première difficulté dans le cadre de la sélection d'attributs est le grand nombre de bras (i.e. actions possibles) dans chaque nœud (i.e. état). La difficulté vient du fait que les MABs sont biaisés vers l'exploration lorsqu'ils sont en présence d'un trop grand nombre de bras [Wang 08]. Or UCT accentue ce biais en utilisant un algorithme de MAB dans chaque nœud du graphe. Cette difficulté est contrée en restreignant et en guidant l'exploration d'UCT. Pour ce faire, deux grands types de solutions sont proposées.

Restriction de l'exploration Deux heuristiques permettent de restreindre l'exploration d'UCT : une heuristique continue et une heuristique discrète. L'*heuristique continue* consiste à choisir une constante d'exploration c_e très faible dans la formule UCB1-tuned (équation 8.3). C'est l'heuristique couramment employée dans le cadre des jeux [Gelly 07, Teytaud 09] même si [Audibert 09] a montré que cette technique est risquée (section 8.1).

L'*heuristique discrète* restreint le nombre de bras considérés dans un nœud F en fonction du nombre T_F de passage dans ce nœud. En pratique, un nouveau bras est considéré à chaque fois que la partie entière de T_F^b est incrémentée, avec $b < 1$ un paramètre de l'algorithme (figure 9.2). Cette heuristique est connue sous le nom d'*élargissement progressif* (PW pour Progressive Widening) et a été proposée simultanément dans [Chaslot 07] et [Coulom 07] où UCT est utilisé pour choisir quel coup jouer au Go (parmi un très grand nombre de coups).

L'avantage de l'heuristique continue est de considérer tous les bras et donc de continuer à explorer quand les premiers bras testés sont « mauvais », alors que l'heuristique discrète est contrainte d'attendre un certain nombre d'itérations avant de pouvoir tester un nouveau bras. Pour ce qui est de l'heuristique discrète, son intérêt est de pouvoir intégrer facilement un critère *a priori* sur l'importance des bras : le premier bras considéré est le meilleur *a priori*, le deuxième bras considéré est le deuxième meilleur *a priori*, ... Comme ce critère *a priori* n'est utilisé que pour ordonner les bras, il n'a pas besoin d'être homogène relativement à la fonction de récompense des feuilles, qui définit l'objectif de maximisation d'UCT. Un exemple d'hybridation entre l'approche discrète de l'élargissement progressif et une heuristique externe est donné dans le cadre de l'apprentissage actif [Rolet 09], où le critère de l'*incertitude maximale* (peu coûteux) est utilisé pour ordonner *a priori* les bras (ici les instances).

Évaluation rapide de la valeur d'une action L'exploration peut bénéficier de connaissances apprises au cours des itérations [Gelly 07]. Dans le cadre du Go [Brügmann 93, Gelly 07], les connaissances apprises au cours des itérations sont résumées par un score appelé RAVE (pour Rapid Action Value Estimation). Ce score associe à chaque action f la récompense moyenne des nœuds terminaux F_t atteints lors des itérations précédentes en utilisant cette action :

$$\text{g-RAVE}(f) = \text{moyenne } \{V(F_t), f \in F_t\} \tag{9.4}$$

Alors que g-RAVE fournit une indication globale de l'intérêt d'une action, il est aussi intéressant de considérer la récompense moyenne conditionnellement au nœud courant F :

$$\ell\text{-RAVE}_F(f) = \text{moyenne } \{V(F_t), F \rightsquigarrow F_t, f \in F_t\} \tag{9.5}$$

où la moyenne est prise sur les itérations passant par le nœud F et terminant au nœud F_t (noté $F \rightsquigarrow F_t$).

Un cas particulier est celui de l'attribut stop. Le score RAVE qui lui est associé dépend de la taille d du sous-ensemble d'attributs auquel il est ajouté. Formellement, si on note $f_s^{(d)}$ l'attribut stop à la profondeur d, g-RAVE $\left(f_s^{(d)}\right)$ et ℓ-RAVE$_F$ $\left(f_s^{(d)}\right)$ sont définis comme la récompense moyenne de tous les nœuds terminaux de taille $d+1$:

$$\text{g-RAVE}\left(f_s^{(d)}\right) = \text{moyenne } \{V(F_t), |F_t| = d+1\} \tag{9.6}$$

$$\ell\text{-RAVE}_F\left(f_s^{(d)}\right) = \text{moyenne } \{V(F_t), F \rightsquigarrow F_t, |F_t| = d+1\} \tag{9.7}$$

Notons que pour ℓ-RAVE$_F$ $\left(f_s^{(d)}\right)$, seul sa valeur en $d = |F|$ sera utilisée. Or cette valeur vérifie :

$$\ell\text{-RAVE}_F\left(f_s^{(|F|)}\right) = \hat{\mu}_F\left(f_s\right)$$

Donc il n'est pas nécessaire de calculer et stocker ℓ-RAVE$_F$ pour l'attribut stop.

g-RAVE, ℓ-RAVE$_F$ et $\hat{\mu}_F$ fournissent tous les trois une estimation de l'intérêt d'une action, i.e. dans notre cas de la pertinence d'un attribut (pour ℓ-RAVE$_F$ et $\hat{\mu}_F$, cette estimation est conditionnée à un nœud particulier de l'arbre : F). Ces trois scores proposent en fait un compromis biais/variance différent. D'un coté, l'information réellement recherchée pour choisir l'action à prendre est celle fournie par $\hat{\mu}_F$; g-RAVE et ℓ-RAVE$_F$ ne font qu'approcher cette information (avec une précision plus importante pour ℓ-RAVE$_F$). D'un autre coté, g-RAVE (respectivement ℓ-RAVE$_F$) se fonde sur plus d'itérations que ℓ-RAVE$_F$ (resp. $\hat{\mu}_F$) et sera donc de variance plus faible.

Guidage de l'exploration Les scores RAVE sont utilisés pour concentrer l'exploration de l'UCT sur les actions les plus prometteuses. L'utilisation de RAVE dépend de l'heuristique choisie pour restreindre l'exploration. Dans le cas de l'heuristique continue, le terme $\hat{\mu}_F$ de l'UCB est remplacé par une somme pondérée de lui-même avec les scores RAVE :

$$(1-\alpha)\cdot\hat{\mu}_F(f) + \alpha\left((1-\beta)\cdot\ell\text{-RAVE}_F(f) + \beta\cdot\text{g-RAVE}(f)\right) + \sqrt{\frac{c_e\log\left(T_F\right)}{t_F(f)}\min\left(\frac{1}{4}, \hat{\sigma}_F^2(f) + \sqrt{\frac{2c_e\log\left(T_F\right)}{t_F(f)}}\right)} \tag{9.8}$$

En s'inspirant de [Gelly 07], les coefficients de pondération sont définis par

$$\alpha = \frac{c}{c + t_F(f)} \tag{9.9}$$

$$\beta = \frac{c_l}{c_l + t_{lF}(f)} \tag{9.10}$$

où c et c_l sont deux paramètres, et $t_{lF}(f)$ représente le nombre d'itérations passant par F et utilisant l'action f.

$$t_{lF}(f) = |\{F_t, F \rightsquigarrow F_t, f \in F_t\}|$$

Ainsi de façon générale, g-RAVE a le plus d'influence lors des premières itérations, puis l'influence se déplace vers ℓ-RAVE$_F$ et enfin vers $\hat{\mu}_F$. Plus particulièrement, ℓ-RAVE$_F$ a plus de poids que g-RAVE (respectivement $\hat{\mu}_F$ a plus de poids que les scores RAVE) à partir du moment où $t_{lF}(f) > c_l$ (resp. $t_F(f) > c$) i.e. à partir du moment où ℓ-RAVE$_F$ (resp. $\hat{\mu}_F$) est fondé sur suffisamment d'itérations.

On peut remarquer que l'arbre de recherche et le score RAVE construits par FUSE inter-agissent. D'une part, RAVE guide l'exploration de FUSE. En effet, RAVE indique les attributs les plus prometteurs et FUSE biaise sont exploration vers ces attributs. D'autre part, FUSE peut être vu comme un moyen sophistiqué de construire RAVE, puisque FUSE choisit les sous-ensembles d'attributs à tester et donc les scores utilisés pour mettre à jour RAVE.

9.2.2 Politique en dehors de l'arbre de recherche : gestion de l'horizon inconnu

Un des objectifs de la sélection d'attributs est de déterminer le nombre d'attributs nécessaires pour avoir la plus petite erreur en généralisation. Le fait que ce nombre d'attributs ne soit pas connu au départ implique que l'UCT ne sait pas *a priori* à quelle profondeur s'arrêter. On est donc face à un problème avec horizon inconnu (même si on sait qu'il est fini puisque l'on ne peut sélectionner au maximum que tous les attributs).

Lors de la phase « bandit » de l'UCT, ce problème est traité en considérant le fait de s'arrêter comme une action équivalente au fait d'ajouter un attribut (concrètement s'arrêter équivaut à ajouter l'attribut stop). Mais dans la phase aléaoire, l'action *je m'arrête* est traitée différemment des autres actions. Si d attributs ont d'ores et déjà été sélectionnés, on s'arrête avec une probabilité $1 - q^d$ où $q < 1$ est un paramètre. Sinon, un attribut pris au hasard est ajouté à l'ensemble des attributs sélectionnés jusque là.

L'arrêt avec une probabilité $1-q^d$ permet de fournir un *a priori* sur le nombre optimal d'attributs tout en laissant l'UCT tester différents nombres d'attributs.

9.2.3 Récompense bon marché

Une récompense bon marché est une condition nécessaire pour l'utilisation d'un algorithme Monte-Carlo tel que l'UCT. En effet UCT demande l'évaluation de la récompense pour un minimum de dix mille itérations ; et dans les expériences ce chiffre s'élève plus volontiers à quelques centaines de milliers d'itérations. C'est pourquoi la récompense se doit d'être rapide à calculer. Mais en même temps, la récompense doit estimer de façon non biaisée l'erreur en généralisation associée à un ensemble d'attributs.

La seconde contribution de l'approche proposée concerne la définition d'une telle estimation. Elle se fonde sur un très faible nombre d'exemples et sur un espace d'hypothèse très simple : la

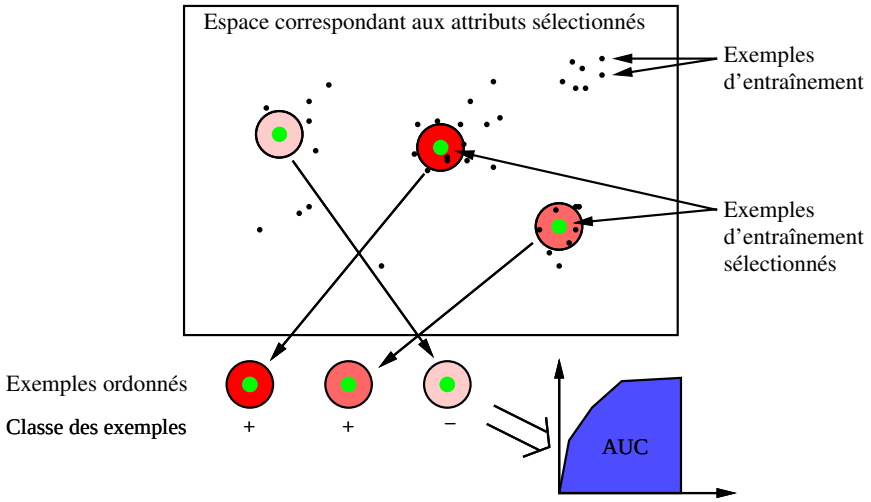


FIG. 9.3 – Calcul de la récompense utilisée par FUSE. Dans l'espace des attributs sélectionnés, la proportion d'exemples positifs est mesurée au voisinage de quelques exemples. Ces quelques exemples sont ordonnées d'après cette proportion et la récompense utilisée est l'aire sous la courbe ROC (AUC) associée à cet ordre sur des exemples.

classification par k-plus proches voisins (k-PPV). Le fait de ne s'intéresser qu'à très peu d'exemples permet d'avoir une récompense rapide à calculer. Mais comme on travaille avec peu d'exemples on ne peut que regarder des propriétés locales autour de ces exemples, d'où l'utilisation d'un k-PPV. Et finalement le k-PPV a l'avantage d'être lui même rapide à calculer.

Concrètement, la récompense associée à un sous-ensemble d'attributs F est définie comme suit (figure 9.3). Soit d_F la distance fondée sur les attributs de F (distance euclidienne pour les attributs numériques et de Hamming pour les attributs catégoriels). Notons $\mathcal{E}$ l'ensemble des exemples d'entraînement et $\mathcal{V}$ un petit sous-ensemble de $\mathcal{E}$. À tout exemple étiqueté $z = (x, y)$ de $\mathcal{V}$ on associe l'ensemble $\mathcal{N}_{F,k}(x)$ de ses k plus proches voisins dans $\mathcal{E}$ d'après d_F ; $s_F(x)$ est alors défini comme le nombre d'exemples positifs parmi ses voisins :

$$s_F(x) = |\{z' \in \mathcal{N}_{F,k}(x),\ y' > 0\}| \tag{9.11}$$

Finalement, la récompense retournée est l'aire sous la courbe ROC (AUC, i.e. critère de Mann Whitney Wilcoxon, algorithme 1) associée à l'ordonnancement des exemples induit par s :

$$V(F) = \frac{|\{(z, z') \in \mathcal{V}^2,\ s_F(x) < s_F(x'),\ y < y'\}|}{|\{(z, z') \in \mathcal{V}^2,\ y < y'\}|} \tag{9.12}$$

Complexité du calcul La récompense est calculée avec une complexité linéaire en la taille n de l'ensemble d'entraînement $\mathcal{E}$, en la taille m de l'ensemble $\mathcal{V}$, et en la taille d du sous-ensemble F d'attributs sélectionnés (à un terme logarithmique près). Plus précisément la complexité est $\tilde{\mathcal{O}}(mnd)$. Cette complexité est raisonnable dans la majorité des cas, mais si le nombre d'exemples d'entraînement devient trop grand, il est possible de se restreindre à un sous-ensemble $\mathcal{E}'$ de $\mathcal{E}$ pour

Algorithm 1 Calcul de l'aire sous la courbre ROC (AUC).

Entrées : s_F, function de score sur les exemples ; $(x_1, y_1), \ldots, (x_m, y_m)$, exemples étiquetés ordonnés par score décroissant ; m^+, nombre d'exemples positifs ; m^-, nombre d'exemples négatifs.

Sorties : $AUC \in [0,1]$, l'Aire sous la courbe ROC associée à l'ordre sur les exemples

1. $(FP, VP, FP_{prec}, VP_{prec}) \leftarrow (0,0,0,0)$
2. $AUC \leftarrow 0$
3. $s_{prec} \leftarrow \infty$
4. **pour** i **de** 1 **à** m **faire**
5. **si** $s_F(x_i) \neq s_{prec}$ **alors**
6. $AUC \leftarrow AUC + \frac{(FP - FP_{prec})(VP + VP_{prec})}{2}$
7. $s_{prec} \leftarrow s_F(x_i)$
8. $FP_{prec} \leftarrow FP$
9. $VP_{prec} \leftarrow VP$
10. **finsi**
11. **si** $y_i > 0$ **alors**
12. $VP \leftarrow VP + 1$
13. **sinon** /* $y_i < 0$ */
14. $FP \leftarrow FP + 1$
15. **finsi**
16. **fin pour**
17. $AUC \leftarrow AUC + \frac{(n^- - FP_{prec})(n^+ + VP_{prec})}{2}$
18. $AUC \leftarrow \frac{AUC}{n^+ n^-}$ /* normalisation */
19. **retourne** AUC

la recherche des k plus proches voisins. Ainsi la complexité devient $\tilde{\mathcal{O}}(mn'd)$ où n' est la taille de $\mathcal{E}'$.

9.2.4 Mise à jour des informations dans l'arbre de recherche

Une fois la récompense calculée, il reste à mettre à jour les informations contenues dans l'arbre de recherche. Dans le cas de la sélection d'attributs, UCT agit sur un graphe. Il y a donc plusieurs chemins qui peuvent potentiellement mener au même nœud final. Dans [de Mesmay 09] où UCT agit aussi sur un graphe, la mise à jour se fait pour tous les nœuds de l'arbre de recherche appartenant à un chemin qui mène au nœud terminal sélectionné.

Mais dans le cas de FUSE, seuls les nœuds du chemin parcouru par l'itération sont mis à jour. En effet, comme pour le Go [Gelly 07] et pour le Havannah [Teytaud 09], il y a un nombre exponentiel de nœuds pouvant mener au nœud final sélectionné. La mise à jour de tous les chemins est donc écartée pour des raisons de coût de calcul.

9.2.5 Déroulement global de FUSE

Resultat FUSE procède par une succession d'itérations (algorithme 2). Ceci permet en particulier à FUSE de tenir compte des contraintes temporelles de l'utilisateur, en fournissant une solution approchée à tout instant. La solution retournée est un sous-ensemble d'attributs pertinents. La procédure UCT standard retourne le chemin obtenu en choisissant en chaque nœud, l'action sélectionnée le plus souvent lors des itérations. Cette stratégie (notée FUSE^{B}par la suite) conduit à un algorithme de type enveloppe. De part son fonctionnement FUSE met en haut du sous-arbre de recherche les attributs les plus informatifs. Il est donc possible d'ordonner les attributs sélectionnés en suivant l'ordre du chemin correspondant.

Mais FUSE nourrit aussi le score RAVE qui estime la pertinence des attributs. Le score RAVE peut ainsi être utilisé comme un critère permettant d'ordonner les attributs, donnant lieu à un algorithme de type filtre. Plus formellement, l'algorithme FUSE^R est paramétré par le nombre d'attributs souhaités, et retourne les ℓ premiers attributs par score RAVE décroissant.

Par la suite, les prefix 'C-' et 'D-' permettront de différencier FUSE utilisant respectivement la stratégie continue et la stratégie discrète pour contrôler l'exploration de l'UCT. Globalement quatre versions différentes de FUSE sont donc proposées (tableau 9.1).

Contrôle de l'exploration \ Sous-ensemble d'attributs retourné (type de stratégie)	Meilleur chemin dans l'arbre de recherche (enveloppe)	Ordonnancement des attributs induit par RAVE (filtre)
Heuristique continue (c_e petit)	C-FUSE^B	C-FUSE^R
Heuristique discrète (élargissement progressif)	D-FUSE^B	D-FUSE^R

TAB. 9.1 – Résumé des 4 versions possibles de l'algorithme proposé : FUSE.

Algorithm 2 Algorithme décrivant FUSE.

Entrées : T, nombre d'iterations ; CE, contrôle de l'exploration
Sorties : $\mathcal{T}$, sous-arbre de recherche ; g-RAVE

$\mathcal{T} \leftarrow \emptyset$
$\forall f$, g-RAVE$(f) \leftarrow 0$
pour t **de** 1 **à** T **faire**
    $(F_t, V) \leftarrow$ Iterate$(\mathcal{T}$, g-RAVE$, \emptyset)$
    Mise à jour de g-RAVE
fin pour

Sous-fonction : Iterate
Entrées : $\mathcal{T}$, sous-arbre de recherche ; g-RAVE ; F, sous-ensemble d'attributs
Sorties : F_t, sous-ensemble d'attributs sélectionné ; V, récompense associée

si F final **alors**
    $V \leftarrow$ Reward$(F \setminus \{f_s\})$
    retourne (F, V)
sinon
    si $T_F \neq 0$ **alors** /* dans le sous-abre de recherche */
        si CE = discret **alors**
            $f^\star \leftarrow \underset{f \in \text{AttributsAutorises}(F)}{\text{argmax}}$ UCB1-tuned(F, f) /* équation 8.3 */
        sinon /* CE = continue */
            $f^\star \leftarrow \underset{f \in \bar{\mathcal{F}} \setminus F}{\text{argmax}}$ compromis UCB/RAVE(F, f) /* équation 9.8 */
        finsi
        $(F_t, V) \leftarrow$ Iterate($\mathcal{T}$, g-RAVE, $F \cup \{f^\star\}$)
        Mise à jour de T_F, $t_F(f)$, $\hat{\mu}_F(f)$, $\hat{\sigma}^2_F(f)$, et ℓ-RAVE$_F(.)$
    sinon /* nœud ajouté */
        $(F_t, V) \leftarrow$ Iterate_random($\mathcal{T}$, g-RAVE, F)
        Mise à jour de T_F et ℓ-RAVE$_F(.)$
    finsi
    retourne (F_t, V)
finsi

Sous-fonction : Iterate_random
Entrées : $\mathcal{T}$, sous-arbre de recherche ; g-RAVE ; F, sous-ensemble d'attributs
Sorties : F_t, sous-ensemble d'attributs sélectionné ; V, récompense associée

tantque $rand() < q^{|F|}$ **faire**
    $f^\star \leftarrow$ attribut tiré uniformément dans $\mathcal{F} \setminus F$
    $F \leftarrow F \cup \{f^\star\}$
fin tantque
$V \leftarrow$ Reward(F)
retourne (F, V)

FUSE ne retourne pas la même information que les autres algorithmes utilisant UCT [Gelly 07, Rolet 09, Teytaud 09]. Ces algorithmes sélectionnent la meilleure action puis relance UCT pour trouver la meilleure deuxième action et ainsi de suite... FUSE, au contraire, sélectionne directement la meilleure séquence d'actions. Cette différence vient du fait que contrairement aux autres algorithmes, FUSE ne reçoit aucune information après avoir choisi la première action. Donc il n'a aucune raison de la choisir définitivement avant de chercher la deuxième. À l'opposé, [Gelly 07, Rolet 09, Teytaud 09] reçoivent une information après avoir sélectionné la première action : le coup joué par l'adversaire [Gelly 07, Teytaud 09] ou l'étiquette de l'exemple sélectionné [Rolet 09]. Ils ont donc tout intérêt à attendre cette information avant de chercher la meilleure deuxième action.

Hypothèse retournée Enfin, une fois les attributs sélectionnés, il reste à apprendre une hypothèse utilisant ces attributs, dont la qualité permettra finalement de valider l'intérêt du sous-ensemble d'attributs choisi. Ainsi que nous l'avons mentionné, le choix d'un critère faisant intervenir un k-PPV a pour seul but de permettre une évaluation rapide, sans présupposé sur la complexité de l'hypothèse pertinente. La seule hypothèse de travail faite est ainsi que la métrique induite par les attributs choisis est localement pertinente du point de vue de la classification.

9.3 Résumé

La sélection d'attributs ne peut s'envisager que sous sa forme enveloppe si elle veut se confronter à des données contenant simultanément des attributs redondants, de nombreux attributs non-pertinents et des attributs conditionnellement pertinents. Or le cadre enveloppe pour la sélection d'attributs nécessite de résoudre un problème d'optimisation, qui est difficile pour deux raisons :

- la fonction dont on recherche le minimum (l'erreur en généralisation) est inconnue et ne peut qu'être approchée ;
- l'ensemble des solutions à tester est un ensemble discret de taille exponentielle (en le nombre d'attributs).

Ce chapitre a introduit notre approche se confrontant à la sélection d'attributs sous ses contraintes fortes : FUSE. FUSE se fonde sur le fait que la sélection d'attributs se réécrit comme un problème d'apprentissage par renforcement sur un processus de décision markovien. La sélection d'attributs peut donc s'envisager comme un jeu à un joueur dont l'objectif est de minimiser l'erreur en généralisation (ou son estimateur).

FUSE étend l'algorithme UCT au cas de la sélection d'attributs. UCT est l'algorithme conduisant aux meilleurs résultats sur plusieurs jeux réputés difficiles (Go, Havannah..). Son application au cadre de la sélection d'attributs a nécessité quelques adaptations :

- une politique pour contrôler le facteur de branchement dans l'arbre de recherche ;
- une politique en dehors de l'arbre de recherche qui permette de gérer l'horizon inconnu, i.e. le fait que FUSE doit déterminer le nombre d'attributs à sélectionner ;
- une récompense qui permette d'estimer (en un temps raisonable) l'erreur en généralisation associée à un sous-ensemble d'attributs.

D'un point de vue théorique, FUSE converge vers la politique optimale, i.e. la politique qui sélectionne le sous-ensemble d'attributs minimisant l'erreur en généralisation. Le chapitre suivant s'intéresse au comportement de FUSE en pratique. Les questions abordées sont notamment la vitesse de convergence de FUSE, ainsi que la comparaison entre les résultats obtenus par FUSE et ceux obtenus par les algorithmes de l'état de l'art. Le chapitre suivant a aussi vocation à déterminer les

avantages et inconvénients des différentes variantes de FUSE proposées.

Chapitre 10

Validation expérimentale

La validation expérimentale considère un ensemble de problèmes tests de l'état de l'art, issu d'un défi en sélection d'attributs (Feature Selection Challenge, NIPS 2003) [Guyon 04], et qui continue de faire référence pour la validation des algorithme de sélection d'attributs.

Cette validation a pour but d'évaluer comparativement FUSE par rapport aux algorithmes existants.

10.1 Protocole expérimental

L'objectif des expériences est de répondre à trois questions :

1. **Quelle est la meilleur variante de FUSE ?** Quatre variantes de FUSE sont proposées (tableau 9.1). Quels sont les avantages et inconvénients de chacune d'entre-elles ?
2. **Dans quelles situations FUSE apporte-t-il un plus ?** Il existe de nombreux algorithmes de sélection d'attributs (chapitre 7). Les expériences doivent permettre d'évaluer les forces et les faiblesses de FUSE par rapport aux autres algorithmes de l'état de l'art, et si possible lier ces forces et faiblesses aux caractéristiques du problème et des données considérées.
3. **Comment évolue la qualité du sous-ensemble d'attributs sélectionné par FUSE en fonction du temps de calcul ?** Ainsi que détaillé au chapitre précédent, FUSE est un algorithme *anytime*, capable à toute itération de retourner une solution. La question est donc de savoir la vitesse de convergence de l'algorithme vers une solution satisfaisante.

Cette section décrit le déroulement des diverses expériences effectuées pour répondre à ces trois questions.

10.1.1 Bases de données

Trois bases de données ont été utilisées pour répondre à ces questions dont les caractéristiques sont détaillées dans le tableau 10.1. Ces trois bases correspondent toutes à la résolution d'un problème d'apprentissage supervisé binaire (séparer les exemples positifs des exemples négatifs) avec

Base de données	Nombre d'exemples	Nombre total d'attributs	Nombre d'attributs pertinents	Propriétés
Madelon	2 600	500	20	Type XOR
Arcene	200	10 000	7 000	Disjonctive
Colon	62	2 000	NF	« Facile »

TAB. 10.1 – Charactéristiques des bases de données.

des attributs numériques. Madelon et Arcene sont deux bases de données incluant des attributs corrélés entre eux. Ces deux bases de données ont été introduites lors du concours de sélection d'attributs qui s'est déroulé pour la conférence NIPS 2003 [Guyon 04]. Colon est une base de données introduites par [Alon 99].

Madelon est une base de données générée artificiellement. Le concept sous-jacent est un ensemble de cliques réparties sur les sommets d'un hyper-cube en dimension 5. Les attributs pertinents sont donc ces 5 attributs *initiaux* ainsi que 15 attributs additionnels : 5 attributs *redondants* correspondant chacun à une combinaison linéaire des attributs initiaux et 10 attributs *répétés* correspondant chacun à une copie d'un attribut initial ou redondant. À ces 20 attributs pertinents sont ajoutés 480 attributs sans lien avec la classe. Le bruit intervient à deux niveaux. D'une part chaque attribut numérique est perturbé par un bruit gaussien. D'autre part, 1% des exemples ont une étiquette altérée.

Et finalement tous les attributs sont bruités et la classe d'1% des exemples est permutée.

L'intérêt de cette base de données est qu'elle contient à la fois des attributs redondants, mais aussi des attributs dont l'intérêt pour la classification est conditionné au fait que l'on utilise d'autres attributs (attributs conditionnellement pertinents).

Arcene est aussi une base de données générée artificiellement, mais à partir de bases de données réelles. Cette base de données est en fait l'union de trois bases de données de détection de cancer (cancer des ovaires dans un cas et cancer de la prostate pour les deux autres cas). Les divers prétraitements appliqués à la base de données sont détaillés dans [Guyon 03a]. Au final, Arcene contient 10 000 attributs dont 3 000 sont non pertinents. Mais pour obtenir de bons résultats en classification, les 7 000 autres attributs ne sont pas tous nécessaires. Pour preuve, les meilleurs résultats jusqu'à aujourd'hui sur cette base de données ont été obtenus par [Neal 06] en utilisant uniquement 100 attributs.

Colon est une base de données couramment utilisée dans la littérature de sélection d'attributs. Elle contient le niveau d'expression de 2 000 gènes pour des cellules cancéreuses ou non. Les meilleurs résultats sur cette base de données sont obtenus habituellement avec un classifieur linéaire. Il s'agit donc d'une base « simple ». Son inclusion dans l'étude correspond au souci de vérifier si FUSE peut également obtenir des résultats raisonnables dans un cas simple.

10.1.2 Performances mesurées

Les résultats fournis correspondent aux résultats moyens obtenus sur cinquante découpages de la base de données en un ensemble d'entraînement contenant 80% des exemples et un ensemble

Expérience	Base de données	C	σ_0^2
Anytime (section 10.3)	Madelon	10	1
Validation croisée (section 10.2)	Madelon	1, 10	0.2, 0.5, 1
	Arcene	1, 10, 100	10, 50, 100, 200, 500, 1 000, 2 000, 5 000, 10 000
	Colon	1,10	0.2, 0.5, 1, 2, 5, 10

TAB. 10.2 – Valeurs testées pour les hyper-paramètres des SVMs. C est le paramètre contrôlant le compromis entre l'erreur empirique et le terme de régularisation. Le noyau utilisé est un noyau gaussien de paramètre d'étalement $\sigma^2 = d\sigma_0^2$, où d est le nombre d'attributs utilisés (i.e. $k_F(x, x') = \exp\left(-\frac{d_F(x,x')}{|F|\sigma_0^2}\right)$).

de test contenant les 20% restants. Plus précisément, les cinquante découpages correspondent à dix validations croisées successives avec cinq blocs[1]. Afin de permettre une comparaison fondée et équitable avec l'état de l'art sur le concours de NIPS 2003, l'algorithme d'apprentissage supervisé choisi est une *machine à vecteurs de support* (SVM) de noyau gaussien.

Plus précisément, pour chaque ensemble d'entraînement $\mathcal{E}_i$, chaque algorithme de sélection d'attributs $\mathcal{A}_j$, fournit un ordre $\pi_{i,j}$ sur les attributs. Pour chaque taille d de sous-ensemble d'attributs, un SVM gaussien est appris en utilisant les d premiers attributs de $\pi_{i,j}$. Les expériences considèrent l'erreur obtenue par ce SVM sur l'ensemble de test associé à $\mathcal{E}_i$, excepté une expérience de la section 10.3 qui s'intéresse à l'évolution du nombre d'attributs sélectionnés par FUSE^B.

Les hyper-paramètres du SVM gaussien sont tous, sauf exception, choisis pour chaque ordre $\pi_{i,j}$ et taille d en effectuant une validation croisée à cinq blocs sur l'ensemble d'entraînement. La seule exception est celle des résultats présentés en section 10.3, qui utilisent des hyper-paramètres fixés une fois pour toutes (tableau 10.2) pour garder un temps de calcul raisonnable.

Suivant les conseils de [Guyon 07, Shen 08], les données de Madelon et Colon sont centrées et normalisées. Naturellement, dans l'intérêt d'une procédure non-biaisée, les paramètres sont appris sur l'ensemble d'entraînement uniquement.

Remarque Notons que la procédure de validation recommandée par [Dietterich 98], i.e. le fait de moyenner les erreurs obtenues sur cinq tirages de validation croisée à deux blocs, n'est pas appropriée dans le cas considéré, en raison du petit nombre d'exemples composant certaines bases de données (en particulier pour Colon qui ne contient que soixante-deux exemples).

10.1.3 Algorithmes de référence

Les performances de FUSE sont comparées aux performances de plusieurs algorithmes de références en terme de sélection d'attributs :

Lasso : proposé par [Tibshirani 94], Lasso est un classifieur linéaire qui vise à minimiser la somme

[1] La validation croisée à K blocs consiste à découper les données en K blocs de taille similaire. Successivement, chaque groupe de $K-1$ blocs est utilisé comme données d'entraînement, et l'hypothèse ainsi apprise est testée sur le bloc restant.

des carrés des erreurs, avec un terme de régularisation se fondant sur la norme L_1 des hyper-paramètres de l'hypothèse considérée.

CFS : la sélection d'attributs fondée sur la corrélation (Correlation-based Feature Selection) [Hall 00] a pour objectif principal d'éliminer les attributs redondants ;

Forêt aléatoire : le score de Gini calculé à partir d'une forêt aléatoire [Breiman 84] permet d'estimer la pertinence d'un attribut conditionnellement à l'utilisation d'autres attributs ;

Un test de cohérence est aussi proposé au travers de l'approche nommée RAND^R. Cette approche consiste à ordonner les attributs selon RAVE. Mais au lieu d'apprendre RAVE à partir de sous-ensembles d'attributs proposés à chaque itération par FUSE, RAND^R apprend RAVE à partir de sous-ensembles d'attributs pris aléatoirement[2].

L'objectif de RAND^R est de vérifier l'impact de FUSE lors de la proposition de sous-ensembles d'attributs. Comme indiqué en section 9.2.1, FUSE peut être vu comme un moyen sophistiqué de choisir les sous-ensembles d'attributs utilisés pour la construction de RAVE. Symétriquement, RAND^R peut être vu comme un moyen « simple » de choisir ces sous-ensembles d'attributs. Ainsi la comparaison entre FUSE^R et RAND^R permet de s'assurer que la sophistication apportée par FUSE améliore la qualité du score RAVE construit.

Enfin, pour des raisons de temps de calcul, FUSE et CFS n'utilisent que les deux mille meilleurs attributs selon ANOVA lors des expériences regardant l'erreur en validation croisée sur Arcene.

10.1.4 Hyper-paramètres de FUSE

Les résultats présentés pour FUSE sont ceux obtenus après 200 000 itérations. Le temps de calcul correspondant sur un Intel Core 2×2.6GHz avec une mémoire vive de 2GB est respectivement de 45 minutes, 5 minutes et 4 minutes pour Madelon, Arcene et Colon (temps sur la base d'entraînement, i.e. sur 80% de la base de données).

Diverses valeurs des hyper-paramètres de FUSE sont testées au cours des expériences. Ces valeurs sont reportées dans le tableau 10.3. Pour des raisons de lisibilité, seuls les résultats des meilleurs d'entre-eux sont présentés.

10.2 Erreur en validation croisée

Cette partie compare FUSE aux différents algorithmes de référence sur les trois bases de données considérées. Les courbes montrent les résultats obtenus par FUSE^R, mais pas ceux obtenus par FUSE^B. En effet l'arbre de recherche n'est pas assez profond et donc les sous-ensembles d'attributs sélectionnés par FUSE^B sont de trop petite taille pour obtenir de bons résultats (section 10.3). De plus, empiriquement, les attributs sélectionnés pas FUSE^B (ainsi que leur ordre) correspondent quasiment à ceux sélectionnés par FUSE^R. Les courbes de résultat de FUSE^B sont donc similaires au début des courbes de résultat de FUSE^R.

Les figures 10.1, 10.2 et 10.3 montrent l'erreur en validation croisée obtenue suivant le nombre d'attributs sélectionnés pour les différents algorithmes de référence et pour FUSE^R après 200 000 itérations. L'erreur est celle obtenue avec un SVM gaussien dont les hyper-paramètres sont choisis après une validation croisée à cinq blocs sur l'ensemble d'entraînement.

[2]La taille des sous-ensembles d'attributs utilisés dépend de la base de données considérée : 20 attributs pour Madelon, 200 pour Arcene et 50 pour Colon.

			Contrôle de l'exploration			
			Heuristique discrète		Heuristique Continue	
Paramètre	k-PPV	q	c_e	b	c_e	c, c_l
Valeur	5-PPV	$1-10^i$	1	1/2	10^i	10^i
i		$\{-1,-3,-5\}$			$\{-4,-2,0,2\}$	$\{-\infty,2,4\}$

TAB. 10.3 – Valeurs testées pour les hyper-paramètres de FUSE. La récompense est fondée sur un espace d'hypothèse de type k plus proches voisins (k-PPV, section 9.2.3). En dehors du sous-arbre de recherche, chaque attribut ajouté l'est avec une probabilité $q^{|F|}$ (section 9.2.2). c_e est la constante d'exploration de la formule UCB1-tuned (equation 8.3) utilisée par l'UCT pour se diriger dans le sous-arbre de recherche (section 8.4). Lors de l'utilisation de l'heuristique discrète pour le contrôle de l'exploration, chaque UCB n'utilise que $\lfloor T_F^b \rfloor$ bras (section 9.2.1). c et c_l contrôlent le compromis entre $\hat{\mu}$ et les divers scores RAVE lors de l'utilisation de l'heuristique continue pour le contrôle de l'exploration (section 9.2.1).

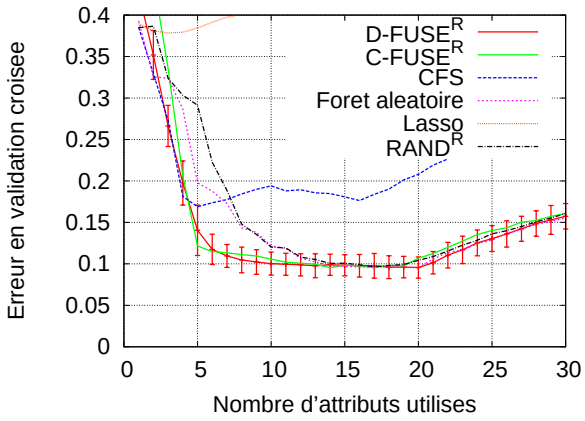


FIG. 10.1 – Erreur en validation croisée sur Madelon en fonction du nombre d'attributs sélectionnés. Les résultats de FUSE et RAND^R sont ceux obtenus après 200 000 itérations. RAND^R utilise le score RAVE obtenu en sélectionnant un sous-ensemble aléatoire de 20 attributs à chaque itération.

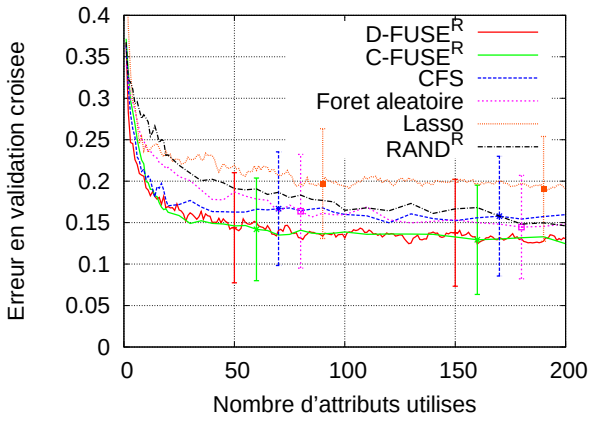


FIG. 10.2 – Erreur en validation croisée sur Arcene en fonction du nombre d'attributs sélectionnés. Les résultats de FUSE et RAND^R sont ceux obtenus après 200 000 itérations. RAND^R utilise le score RAVE obtenu en sélectionnant un sous-ensemble aléatoire de 200 attributs à chaque itération. CFS, FUSE et RAND^R sont lancés sur les 2 000 meilleurs attributs d'après ANOVA.

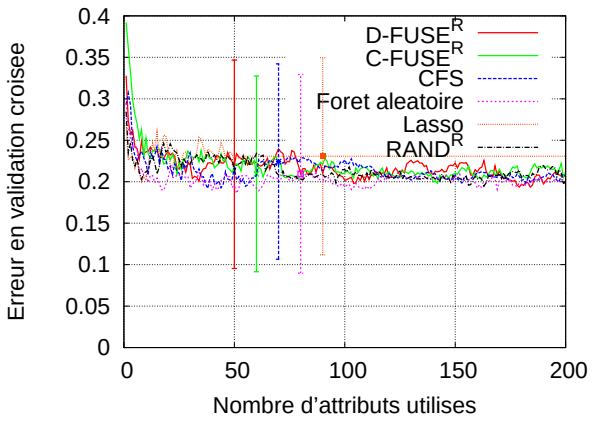


FIG. 10.3 – Erreur en validation croisée sur Colon en fonction du nombre d'attributs sélectionnés. Les résultats de FUSE et RAND^R sont ceux obtenus après 200 000 itérations. RAND^R utilise le score RAVE obtenu en sélectionnant un sous-ensemble aléatoire de 50 attributs à chaque itération.

Le meilleur des deux mondes Sur Madelon et sur Arcene, lorsqu'il s'agit de sélectionner peu d'attributs, FUSE obtient les même résultats que CFS et des résultats significativement meilleurs que les forêts aléatoires. Ainsi, FUSE est capable de détecter les attributs redondants et de ne pas les sélectionner.

Symétriquement, lorsqu'il s'agit de sélectionner beaucoup d'attributs, FUSE obtient les même résultats que les forêts aléatoires et des résultats significativement meilleurs que CFS. Ces résultats indiquent que FUSE, comme les forêts aléatoires, est en mesure de détecter les attributs conditionnellement pertinents.

Enfin, FUSE atteint les meilleurs résultats avec moins d'attributs que les forêts aléatoires. Cela tient encore une fois au fait que FUSE, contrairement aux forêts aléatoires, est capable de détecter et éliminer les attributs redondants.

Globalement, FUSE est donc le seul algorithme apte à gérer correctement à la fois les attributs redondants et les attributs conditionnellement pertinents.

Linéaire / non-linéaire Les résultats de Lasso sur Madelon et Arcene peuvent surprendre étant donnée la capacité de Lasso à fournir ordinairement de très bons résultats. L'explication tient à la nature des concepts sous-jacents à Madelon et Arcene, qui ne sont pas linéairement séparables. Cette non-séparabilité est claire pour Madelon de par la manière dont la base de données a été générée. Pour ce qui est d'Arcene, le meilleur moyen de se convaincre de sa non-séparabilité est de remarquer que les meilleurs résultats obtenus sur cette base de données utilisent un réseau de neurones avec deux couches cachées [Neal 06], i.e. un modèle typiquement non-linéaire.

Sur Colon, en revanche, toutes les approches sont équivalentes. La particularité de Colon est d'être une base de données linéairement séparable.

En conclusion, FUSE donne des résultats équivalents aux autres approches dans les cas « simples » (données linéairement séparables) et FUSE apporte un plus dans les cas de bases de données non linéairement séparables et comportant à la fois des attributs corrélés et des attributs conditionnellement pertinents. Le bon comportement de FUSE lorsque l'on mélange différents facteurs de complexité provient du fait que FUSE teste directement l'efficacité des sous-ensembles d'attributs. Dans le même temps, l'évaluation de nombreux sous-ensembles d'attributs comme une méthode enveloppe, mais en considérant de très nombreux candidats. Cette exploration est rendue possible par la procédure d'évaluation elle-même (section 9.2.3) qui fait intervenir une très petite fraction de la base d'apprentissage.

Contrôle de l'exploration : heuristique continue ou discrète ? Ces résultats permettent aussi de comparer les deux heuristiques de contrôle de l'exploration. Le tableau 10.4 indique les hyper -paramètres conduisant aux meilleurs résultats. C-FUSE et D-FUSE conduisent à des résultats similaires lorsque l'on utilise les meilleurs hyper-paramètres ; mais D-FUSE ne demande que de fixer la probabilité q d'arrêter d'ajouter des attributs (section 9.2.2) alors que C-FUSE demande sur Madelon de bien choisir aussi c_e, c et c_l. D-FUSE s'avère donc d'usage plus simple et plus robuste.

10.3 Un algorithme “anytime”

FUSE est un algorithme *anytime*, i.e. il est en mesure de fournir une solution à tout instant. Cependant, la qualité de cette solution s'accroît au cours du temps, i.e. au cours des itérations.

		Contrôle de l'exploration			
		Heuristique discrète		Heuristique continue	
Paramètre	q	c_e	b	c_e	c, c_l
Valeurs testées i	$1-10^i$ $\{-1,-3,-5\}$	1	1/2	10^i $\{-4,-2,0,2\}$	10^i $\{-\infty,2,4\}$
Madelon	$1-10^{-3}$	1	1/2	×	×
	$1-10^{-1}$	×	×	10^{-2}	$\{(10^2,0),(10^4,0)\}$
Arcene	$1-10^{-1}$	1	1/2	×	×
	$1-10^{-1}$	×	×	10^{-2}	Toutes
	$1-10^{-3}$	×	×	10^{-4}	Presque toutes
Colon	$1-10^{-5}$	1	1/2	×	×
	$1-10^{-5}$	×	×	Toutes	Presque toutes

TAB. 10.4 – Hyper-paramètres de FUSE donnant les meilleurs résultats lors des expériences en validation croisée sur les bases de données Madelon, Arcene et Colon. En dehors du sous-arbre de recherche, chaque attribut ajouté l'est avec une probabilité $q^{|F|}$ (section 9.2.2). c_e est la constante d'exploration de la formule UCB1-tuned (equation 8.3) utilisée par l'UCT pour se diriger dans le sous-arbre de recherche (section 8.4). Lors de l'utilisation de l'heuristique discrète pour le contrôle de l'exploration, chaque UCB n'utilise que $\lfloor T_F^b \rfloor$ bras (section 9.2.1). c et c_l contrôlent le compromis entre $\hat{\mu}$ et les divers scores RAVE lors de l'utilisation de l'heuristique continue pour le contrôle de l'exploration (section 9.2.1).

Cette section s'intéresse à cette évolution au cours du temps, et met en évidence un inconvénient de la version FUSE^{B}de l'algorithme.

La figure 10.4 montre l'évolution au cours des itérations de l'erreur en validation croisée des différentes versions de FUSE ainsi que de RAND^R (FUSE^R et RAND^R sélectionnent les vingt meilleurs attributs d'après RAVE.). Cette courbe correspond à l'erreur obtenue sur MADELON avec un SVM gaussien[3] suivant l'itération à laquelle l'algorithme est arrêté.

Une première remarque est que la version FUSE^B n'obtient pas les meilleurs résultats au bout des 200 000 itérations. L'explication de ces résultats est fournie par la figure 10.5 qui montre l'évolution au cours des itérations du nombre d'attributs sélectionnés par FUSE^B. Pour avoir de bons résultats, il faut que l'algorithme détecte les vingt attributs pertinents. Or FUSE^B détecte au mieux une dizaine d'attributs pertinents. Le problème est donc que le meilleur chemin dans l'arbre de recherche construit par l'UCT est trop petit, i.e. l'arbre de recherche n'est pas assez profond. En somme, les différentes stratégies pour restreindre l'exploration ne sont pas assez agressives.

En revanche, les versions FUSE^R convergent 10 fois plus rapidement que RAND^R. Cela démontre que l'UCT apporte quelque chose à FUSE. Ainsi, même si l'UCT n'est pas en mesure de construire un arbre de recherche suffisamment profond, il est capable de biaiser la recherche dans la bonne direction, et par conséquent de fournir une information plus pertinente par l'intermédiaire des scores RAVE.

[3]Les paramètres du SVM sont fixés à $C = 10$ et $\sigma^2 = d$ où d est le nombre d'attributs sélectionnés.

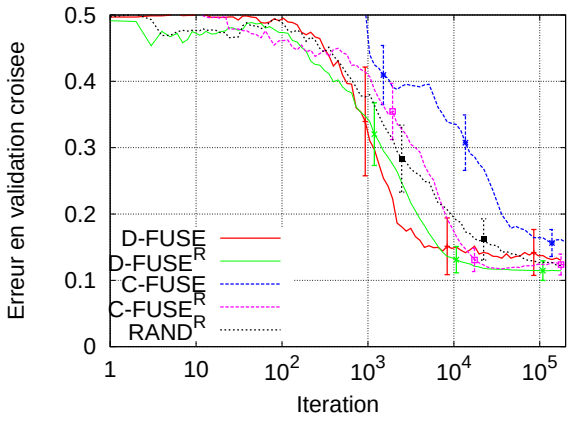


FIG. 10.4 – Erreur en validation croisée sur Madelon en fonction du nombre d'itérations effectuées (échelle logarithmique). RAND^R utilise le score RAVE obtenu en sélectionnant un sous-ensemble aléatoire de 20 attributs à chaque itération.

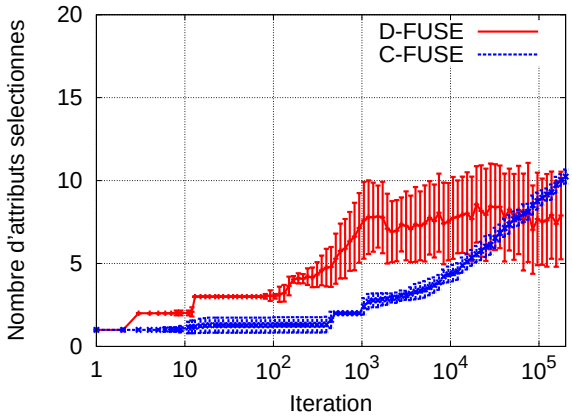


FIG. 10.5 – Nombre d'attributs sélectionnés sur Madelon en fonction du nombre d'itérations effectuées (échelle logarithmique).

Base de données	Algorithme	Erreur en test	Nombe d'attributs utilisés	Nombre d'attributs non-pertinents
Madelon	Obaid Malik	6.17% (1^{st})	20	0
	FSPP2 [Shen 08]	6.22% (2^{nd})	12	0
	Bayes-nn-DFT-combo [Neal 06]	6.22% (13^{th})	500	480
	D-FUSE^R	6.50% (24^{th})	18	0
Arcene	Bayes-nn-red [Neal 06]	7.20% (1^{st})	100	0
	Obaid Malik	8.19% (2^{nd})	1 300	47
	D-FUSE^R (sur tous)	8.42% (3^{rd})	500	34
	D-FUSE^R(sur 2 000)	9.42% (8^{th})	500	0

TAB. 10.5 – Résultats indiqués par le site internet du concours de sélection d'attributs qui a eu lieu à l'occasion de NIPS 2003. Le site retourne l'erreur en test ainsi que le nombre d'attributs non-pertinents parmi les attributs sélectionnés. Sur Arcene, FUSE a été utilisé pour chercher les meilleurs attributs parmi tous les attributs ainsi que pour chercher les meilleurs attributs parmi les 2 000 meilleurs d'après ANOVA.

10.4 Erreur sur les données-test du concours

Cette section fournit les résultats obtenus par FUSE sur le site du concours de sélection d'attributs qui a eu lieu à l'occasion de la conférence NIPS en 2003 [Guyon 04]. Ce site fonctionne toujours, et il est possible d'y tester une approche sur une base de test, i.e. une base de données pour laquelle les étiquettes des exemples n'ont jamais été fournies par les organisateurs du concours. Le site retourne alors l'erreur sur cette base de test ainsi que le nombre d'attributs non-pertinents parmi ceux utilisés pour apprendre le modèle final.

Le tableau 10.5 résume les résultats obtenus par FUSE ainsi que les meilleurs soumissions actuelles d'après le site du concours. La première remarque est que FUSE élimine les attributs non-pertinents (dernière colonne du tableau).

Sur Madelon, FUSE est classé $24^{ème}$ pour l'erreur en test, alors que l'écart entre FUSE et la plus petite erreur en test obtenue est de 0,33% seulement (ce qui correspond à six exemples mal classés en plus). Ainsi FUSE obtient une bonne erreur en test et son classement est principalement dû aux nombreuses approches *ex æquo*.

Sur Arcene, on observe le phénomène inverse : FUSE est bien classé ($3^{ème}$) alors que l'écart entre FUSE et la plus petite erreur en test obtenue est de 1,22%. Cela s'explique par la difficulté de la base de données Arcene, sur laquelle personne n'arrive à égaler les résultats de [Neal 06]. Il est à noter que ces résultats sont obtenus en fournissant à FUSE l'ensemble des attributs, contrairement aux résultats de la section 10.2 qui correspondent à l'utilisation de FUSE sur les deux milles meilleurs attributs selon ANOVA.

En résumer, FUSE est à la fois capable d'éliminer les attributs non-pertinents et de concerver les attributs les plus pertinents, permettant ainsi d'avoir une faible erreur en test.

10.5 Résumé

Ce chapitre s'est intéressé au comportement de FUSE en pratique, i.e. sur des bases de données issue du concours de sélection d'attributs qui c'est déroulé à l'occasion de NIPS 2003. Les différentes variantes de FUSE ont été comparées entre-elles et avec les algorithmes de l'état de l'art.

Il ressort de cette étude, que FUSE est en mesure de traiter correctement les attributs redondants et les attributs conditionnellement pertinents ; alors que les algorithmes de l'état de l'art commettent des erreurs pour l'un ou l'autre de ces deux types d'attributs.

L'inconvénient de FUSE est le temps de calcul nécessaire au traitement d'une base de données. Pour Madelon, par exemple, FUSE ordonne les attributs en quarante cinq minutes, alors que les forêts aléatoire et CFS n'ont besoin respectivement que de cinq minutes et une minute.

Les expériences permettent aussi de déterminer la meilleur variante de FUSE. D'une part, même si la manière dont est contrôlé le facteur de branchement dans l'arbre de recherche n'influe pas sur les meilleurs résultats obtenus, il est plus facile de déterminer les meilleurs paramètres de FUSE lorsque c'est l'approche fondée sur l'élargissement progressif qui est utilisée.

D'autre part, les meilleurs résultats sont obtenus en ordonnant les attributs d'après RAVE, et non pas en utilisant la meilleur branche de l'arbre de recherche construit par FUSE. La raison principale de cette remarque est que l'arbre de recherche n'est pas assez profond pour sélectionner ou ordonner suffisamment d'atributs. Toutefois, l'arbre de recherche de FUSE est déterminant dans sa réussite. En effet, les expériences comparant FUSE^R à RAND^R indiquent que FUSE permet de construire plus rapidement un vecteur RAVE de bonne qualité.

Chapitre 11

Conclusion et perspectives sur la sélection d'attributs

La contribution de ce deuxième chapitre est double. Nous avons tout d'abord reformulé la sélection d'attributs en terme d'apprentissage par renforcement.

Cette nouvelle approche à le mérite de poser le problème en toute généralité, en tenant compte des divers facteurs de difficulté de la sélection d'attibuts que sont la présence d'attributs redondants et celle d'attributs conditionnellement pertinents. Certes, cette formulation n'est pas calculable en un temps raisonnable, pour deux raisons, d'une part la taille de l'espace d'état, d'autre part la donnée de la fonction de récompense.

La seconde contribution du travail présenté est ainsi d'avoir proposé deux approximations, relatives à l'estimation de la fonction récompense et l'exploration de l'espace d'états, se fondant sur une approche de type Monte-Carlo tree search. L'algorithme finalement obtenu, FUSE, étend l'approche UCT à la résolution du problème de sélection d'attributs vu comme un apprentissage par renforcement. Cet algorithme a l'avantage de pouvoir être arrêté à tout moment, tout en conduisant aux résultats de l'état de l'art sur des bases de données du concours de sélection d'attributs qui s'est déroulé à l'occasion de NIPS 2003.

Ces bons résultats ont été obtenus notamment grâce à la récompense utilisée par FUSE. Cette récompense permet une évaluation rapide d'un ensemble d'attributs sans faire d'hypothèse sur la complexité du concept à apprendre. Parallèlement, l'évaluation de nombreux sous-ensembles d'attributs est rendue possible car seule une toute petite partie des exemples est considérée à chaque itération.

Perspectives

Une question reste toujours ouverte : *FUSE doit-il être utilisé dans son mode enveloppe ou filtre ?* La version actuelle de FUSE n'est pas en mesure de construire un arbre de recherche suffisamment profond, mais UCT laisse libre champ à de nombreuses adaptations. Des heuristiques telles que

celles utilisées pour biaiser la sélection des actions en dehors de l'arbre de recherche [Rimmel 10] pourraient accélérer la convergence de l'UCT et permettre ainsi d'obtenir des arbres de recherche plus profonds.

De même, d'autres types de récompenses pourraient être expérimentées et plus particulièrement des récompenses qui seraient évaluées à plusieurs niveaux dans l'arbre, réduisant ainsi la dépendance de FUSE face au paramètre q contrôlant la profondeur moyenne atteinte (section 9.2.2).

Il reste aussi à adapter FUSE au cas des problèmes à plus de deux classes et aux problèmes incluant des attributs à la fois réels et discrets.

Enfin, une perspective intéressante est la question de l'adaptation de FUSE au cas de la construction d'attributs. [de Mesmay 09] utilise un UCT pour rechercher un mot optimal parmi les mots produits par une grammaire. Or, dans certains contexte, les attributs possibles correspondent à des mots d'une grammaire, comme par exemple dans le cas de la construction d'attributs audio [Mierswa 05] ou de règles logiques [Landwehr 07]. Dans ce cadre, UCT est un outil prometteur pour rechercher le ou les meilleurs mots, i.e. le ou les meilleurs attributs.

Chapitre 12

Conclusion et perspectives générales

La question fondamentale abordée dans cette thèse est celle de la représentation des problèmes d'apprentissage et du lien entre la représentation et l'espace des hypothèses.

La première contribution est de démontrer que les limites fondamentales des représentations relationnelles, qui avaient été mises en évidence dans la littérature, n'étaient pas résolues de fait par l'astuce des noyaux. De manière précise, nous démontrons l'existence d'une région d'échec dans le cas des données à instances multiples en montrant que les noyaux-somme permettent d'approcher des concepts moyens mais non des concepts existentiels. La généralité de ce résultat, indépendant de l'algorithme appliqué à la suite de la transformation, repose sur une borne inférieure concernant l'erreur en généralisation, qui a été démontrée dans le chapitre 4 et dont les implications ont été démontrées empiriquement dans le chapitre 5.

Dans une deuxième partie, cette thèse s'est intéressée à un aspect central de la représentation : la sélection d'attributs. De même que dans la première partie, le critère fondamental est de minimiser l'erreur en généralisation et nous montrons que du point de vue de ce critère, le problème de la sélection d'attributs définit un problème d'apprentissage par renforcement. Ce problème est doublement intraitable, en raison de l'erreur en généralisation qui ne peut qu'être estimée, mais aussi en raison de la nature d'optimisation combinatoire de la sélection d'attributs dans ce contexte.

Cependant, le fait d'avoir pu formaliser la sélection d'attributs comme un problème d'apprentissage par renforcement nous a permis d'approcher la solution optimale. Cette approximation repose sur deux principes : le premier consiste en l'exploration du treillis des sous-ensembles d'attributs en utilisant un algorithme de type *Monte-Carlo Tree Search*, et le second est une estimation de l'erreur en généralisation qui est computationnellement frugale sans pour autant se limiter à la considération de modèles linéaires. Ce résultat doit être évalué en tenant compte de sa principale faiblesse, son coût de calcul élevé ; il faut cependant souligner que cette approche reposant sur un principe nouveau nous a permis d'obtenir les résultats de l'état de l'art sur un domaine étudié depuis les débuts de l'apprentissage statistique.

Une des perspectives majeures ouvertes par FUSE, dans la droite ligne de la recherche d'une représentation adéquate, concerne la construction d'attributs [Lim 07, Arai 09]. Comme discuté

dans le chapitre 8, les approches MCTS peuvent être adaptés a la recherche dans un graphe plutôt que dans un arbre, comme par exemple dans le cas des formules d'une grammaire donnée. Dans le cas particulier d'une recherche relationnelle, le plus grand défi consiste à pouvoir explorer l'ensemble des attributs implicitement contenus dans une base de données relationnelles. Ce défi correspond au fait d'éviter d'une part l'écueil d'une mise à plat préalable, sachant qu'une mise à plat ne passe pas à l'échelle en raison du nombre d'attributs à construire, et le fait de traiter directement une représentation en logique du premier ordre, tel que le fait la programmation logique inductive (ILP) [Muggleton 91, Landwehr 07]. Le passage à l'échelle des techniques inductives sur les grandes bases de données, telles que celles issues de la gestion de relations clients, pose en effet des problèmes insurmontables.

Nous avons eu l'occasion, en relation avec Vincent Lemaire et Raphaël Féraud de Orange Labs, de procéder à des études préliminaires sur la faisabilité de l'extension de FUSE à la construction d'attributs sur une base de données en étoile. La validation en vraie grandeur des premiers résultats encourageants obtenus constitue la perspective la plus importante à court terme.

Une perspective de recherche à plus long terme consiste à entrelacer la recherche des exemples les plus informatifs avec la recherche des attributs pertinents. Lorsque le nombre d'exemples est trop important pour pouvoir tous les considérer, la sélection d'exemples cherche à sélectionner les exemples les plus informatifs, afin d'obtenir l'erreur en généralisation la plus faible possible [Rolet 09, Feraud 10]. Une recherche simultanée des exemples informatifs et des attributs pertinents permettrait de traiter les cas où le sous-ensemble d'attributs pertinents dépend du groupe d'exemples considéré.

Liste des figures

Liste des tableaux

Liste des algorithmes

Bibliographie

[Achlioptas 01] Dimitris Achlioptas. *Lower bounds for random 3-SAT via differential equations.* Theoretical Computer Science, vol. 265, no. 1_2, pages 159–185, 2001.

[Achlioptas 06] Dimitris Achlioptas & Federico Ricci-Tersenghi. *On the solution-space geometry of random Constraint Satisfaction Problems.* In Proceedings of the 38^{th} annual ACM symposium on Theory of computing, pages 130–139, New York, NY, USA, 2006. ACM.

[Alon 99] U. Alon, N. Barkai, D. A. Notterman, K. Gishdagger, S. Ybarradagger, D. Mackdagger & A. J. Levine. *Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays.* In Proceedings of the National Academy of Science USA, volume 96, pages 6745–6750, June 1999.

[Alphonse 02] Érick Alphonse & Nicolas Stroppa. *Filtering multi-instance problems for feature selection in ilp.* In Proceedings of the 12^{th} International Conference on Inductive Logic Programming, Work-in-progress trac, 2002.

[Alphonse 08a] Erick Alphonse & Aomar Osmani. *A Model to Study Phase Transition and Plateaus in Relational Learning.* In Proceedings of the 18^{th} international conference on Inductive Logic Programming, pages 6–23, Berlin, Heidelberg, 2008. Springer-Verlag.

[Alphonse 08b] Erick Alphonse & Aomar Osmani. *On the connection between the Phase Transition of the covering test and the learning success rate in ILP.* Machine Learning, vol. 70, no. 2-3, pages 135–150, 2008.

[Anglano 98] Cosimo Anglano, Attilio Giordana, Giuseppe Lo Bello & Lorenza Saitta. *An Experimental Evaluation of Coevolutive Concept Learning.* In Proceedings of the 15^{th} International Conference on Machine Learning, pages 19–27. Morgan Kaufmann, 1998.

[Arai 09] Hiromi Arai, Satoru Watanabe, Takanori Kigawa & Masayuki Yamamura1. *A new modeling method in feature construction for the HSQC spectra screening problem.* Bioinformatics, vol. 25, no. 7, pages 948–953, 2009.

[Audibert 09] Jean-Yves Audibert, Rémi Munos & Csaba Szepesvári. *Exploration-Exploitation tradeoff using variance estimates in Multi-Armed Bandits.* Theoretical Computer Science, vol. 410, no. 19, pages 1876–1902, 2009.

[Auer 02] Peter Auer, Nicolò Cesa-Bianchi & Paul Fischer. *Finite-time Analysis of the Multiarmed Bandit Problem.* Machine Learning, vol. 47, no. 2-3, pages 235–256, 2002.

[Bach 04] Francis R. Bach, Gert R. G. Lanckriet & Michael I. Jordan. *Multiple Kernel Learning, conic duality, and the SMO algorithm.* In Proceedings of the 21^{st} International Conference on Machine Learning, page 6, New York, NY, USA, 2004. ACM.

[Bach 08] Francis Bach. *Exploring large feature spaces with Hierarchical Multiple Kernel Learning.* In Proceedings of the 22^{nd} Annual Conference on Neural Information Processing Systems, pages 105–112, 2008.

[Baskiotis 04] Nicolas Baskiotis & Michèle Sebag. *C4.5 competence map : a Phase Transition-inspired approach.* In Carla E. Brodley, editeur, Proceedings of the 21^{st} International Conference on Machine Learning, volume 69 of *ACM International Conference Proceeding Series*. ACM, 2004.

[Bellman 57] Richard Ernest Bellman. Dynamic programming. Princeton Univ. Press, 1957.

[Berry 85] D. Berry & B. Fristedt. Bandit problems. Chapman and Hall, 1985.

[Bi 05] Jinbo Bi, Yixin Chen & James Z. Wang. *A Sparse Support Vector Machine Approach to Region-Based Image Categorization.* In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, volume 1, pages 1121–1128, Washington, DC, USA, 2005. IEEE Computer Society.

[Bianchetti 02] Jacques Ales Bianchetti, Céline Rouveirol & Michèle Sebag. *Constraint-based Learning of Long Relational Concepts.* In Proceedings of the 19^{th} International Conference on Machine Learning, pages 35–42, 2002.

[Bishop 95] Christopher M. Bishop. Neural Networks for Pattern Recognition. Oxford University Press, Inc., New York, NY, USA, 1995.

[Blum 97] Avrim L. Blum & Pat Langley. *Selection of relevant features and examples in machine learning.* Artificial Intelligence, vol. 97, no. 1-2, pages 245–271, 1997.

[Boser 92] Bernhard E. Boser, Isabelle M. Guyon & Vladimir N. Vapnik. *A training algorithm for optimal margin classifiers.* In Proceedings of the 5^{th} annual workshop on Computational learning theory, pages 144–152, New York, NY, USA, 1992. ACM.

[Botta 93] Marco Botta & Attilio Giordana. *SMART+ : A Multi-Strategy Learning Tool.* In Proceedings of the 13^{th} International Joint Conference on Artificial Intelligence, pages 937–945, 1993.

[Botta 03] Marco Botta, Attilio Giordana, Lorenza Saitta & Michèle Sebag. *Relational Learning as search in a critical region.* Journal of Machine Learning Research, vol. 4, pages 431–463, 2003.

[Boullé 07] Marc Boullé. *Compression-based averaging of selective Naive Bayes classifiers.* Journal of Machine Learning Research, vol. 8, pages 1659–1685, 2007.

[Bradley 98] Paul S. Bradley & O. L. Mangasarian. *Feature Selection via Concave Minimization and Support Vector Machines.* In Proceedings of the 15^{th} International Conference on Machine Learning, pages 82–90, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[Braunstein 05] Alfredo Braunstein, Marc Mézard & Riccardo Zecchina. *Survey propagation : An algorithm for satisfiability.* Random Structures & Algorithms, vol. 27, no. 2, pages 201–226, 2005.

[Breiman 84] Leo Breiman, J. Friedman, C. J. Stone & R.A. Olshen. Classification and Regression Trees. Taylor & Francis, Inc., 1984.

[Brügmann 93] Bernd Brügmann. *Monte Carlo Go*, 1993.

[Bubeck 10a] Sébastien Bubeck. *Bandits Games and Clustering Foundations.* PhD thesis, Université Lille 1, 2010.

[Bubeck 10b] Sébastien Bubeck & Rémi Munos. *Open Loop Optimistic Planning.* In Proceedings of the 23^{rd} Annual Conference on Learning Theory, 2010.

[Chan 07] Antoni B. Chan, Nuno Vasconcelos & Gert R. G. Lanckriet. *Direct convex relaxations of sparse SVM.* In Proceedings of the 24^{th} International Conference on Machine Learning, pages 145–153, New York, NY, USA, 2007. ACM.

[Chaslot 06] Guillaume Chaslot, Jahn-Takeshi Saito, Bruno Bouzy, Jos W. H. M. Uiterwijk & H. Jaap van den Herik. *Monte-Carlo Strategies for Computer Go.* In Pierre-Yves Schobbens, Wim Vanhoof & Gabriel Schwanen, editeurs, Proceedings of the 18^{th} BeNeLux Conference on Artificial Intelligence, pages 83–91, 2006.

[Chaslot 07] Guillaume M. J-B. Chaslot, Mark H. M. Winands, Jos W. H. M. Uiterwijk, H. Jaap van den Herik & Bruno Bouzy. *Progressive Strategies for Monte-Carlo Tree Search.* In P. Wang*et al.*, editeurs, Proceedings of the 10^{th} Joint Conference on Information Sciences, pages 655–661. World Scientific Publishing Co. Pte. Ltd., 2007.

[Cheeseman 91] Peter Cheeseman, Bob Kanefsky & William M. Taylor. *Where the really hard problems are.* In Proceedings of the 12^{th} International Joint Conference on Artificial Intelligence, pages 331–337, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[Chen 89] Sheng Chen, S. A. Billings & W. Luo. *Orthogonal least squares methods and their application to non-linear system identification.* International Journal of Control, vol. 50, pages 1873–1896, 1989.

[Chen 04] Yixin Chen & James Z. Wang. *Image Categorization by Learning and Reasoning with Regions.* Journal of Machine Learning Research, vol. 5, pages 913–939, 2004.

[Chen 06] Yixin Chen, Jinbo Bi & James Z. Wang. *MILES : Multiple-Instance Learning via Embedded Instance Selection.* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 28, no. 12, pages 1931–1947, 2006.

[Cocco 01] S. Cocco & R. Monasson. *Trajectories in phase diagrams, growth processes, and computational complexity : how search algorithms solve the 3-satisfiability problem.* Physical review letters, vol. 86, no. 8, pages 1654–1657, 2001.

[Collobert 02] Ronan Collobert, Samy Bengio & Johnny Mariéthoz. *Torch : A Modular Machine Learning Software Library.* Rapport technique, IDIAP, 2002.

[Cook 71] Stephen A. Cook. *The complexity of theorem-proving procedures.* In Proceedings of the third annual ACM symposium on Theory of computing, pages 151–158, New York, NY, USA, 1971. ACM.

[Coulom 06] Rémi Coulom. *Efficient selectivity and backup operators in Monte-Carlo tree search.* In Computers and Games, pages 72–83, 2006.

[Coulom 07] Rémi Coulom. *Computing Elo Ratings of Move Patterns in the Game of Go.* In Computer Games Workshop, 2007.

[de Mesmay 09] Frédéric de Mesmay, Arpad Rimmel, Yevgen Voronenko & Markus Püschel. *Bandit-based optimization on graphs with application to library performance tuning.* In Proceedings of the 26^{th} International Conference on Machine Learning, pages 729–736, 2009.

[Debnath 91] Asim Kumar Debnath, Rosa L. Lopez de Compadre, Gargi Debnath, Alan J. Shusterman & Corwin Hansch. *Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity.* Journal of Medicinal Chemistry, vol. 34, no. 2, pages 786–797, 1991.

[Dietterich 97] Thomas G. Dietterich, Richard H. Lathrop & Tomás Lozano-Pérez. *Solving the Multiple Instance problem with axis-parallel rectangles.* Artificial Intelligence, vol. 89, no. 1-2, pages 31–71, 1997.

[Dietterich 98] Thomas G. Dietterich. *Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms.* Neural Computation, vol. 10, pages 1895–1923, 1998.

[Dubois 00] Olivier Dubois, Yacine Boufkhad & Jacques Mandler. *Typical random 3-SAT formulae and the satisfiability threshold.* In Proceedings of the 11^{th} annual ACM-SIAM symposium on Discrete algorithms, pages 126–127, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

[Dubois 01] Olivier Dubois & Gilles Dequen. *A backbone-search heuristic for efficient solving of hard 3-SAT formulae.* In Proceedings of the 17^{th} International Joint Conference on Artificial Intelligence, pages 248–253, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[Efron 04] Bradley Efron, Trevor Hastie, Iain Johnstone & Robert Tibshirani. *Least angle regression.* Annals of Statistics, vol. 32, pages 407–499, 2004.

[Feraud 10] Raphaël Feraud, Marc Boullé, Fabrice Clérot, Françoise Fessant & Vincent Lemaire. *The Orange Customer Analysis Platform.* In Proceedings of the 10^{th} International Conference on Data Mining, volume 6171 of *Lecture Notes in Computer Science*, pages 584–594. Springer, 2010.

[Forman 03] George Forman. *An extensive empirical study of feature selection metrics for text classification.* Journal of Machine Learning Research, vol. 3, pages 1289–1305, 2003.

[Fung 00] Glenn Fung & Olvi L. Mangasarian. *Data Selection for Support Vector Machine Classifiers.* In Proceedings of the sixth ACM SIGKDD international conference on Knowledge Discovery and Data mining, pages 64–70, New York, NY, USA, 2000. ACM.

[Furey 00] Terrence S. Furey, Nello Cristianini, Nigel Duffy, David W & David Haussler. *Support Vector Machine Classification and Validation of Cancer Tissue Samples Using Microarray Expression Data.* Bioinformatics, vol. 16, 2000.

[Gärtner 02] Thomas Gärtner, Peter A. Flach, Adam Kowalczyk & Alex J. Smola. *Multi-Instance Kernels.* In Proceedings of the 19^{th} International Conference on Ma-

chine Learning, pages 179–186, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[Gelly 06] Sylvain Gelly, Yizao Wang, Rémi Munos & Olivier Teytaud. *Modification of UCT with Patterns in Monte-Carlo Go*. Rapport de recherche INRIA RR-6062, INRIA, 2006.

[Gelly 07] Sylvain Gelly & David Silver. *Combining online and offline knowledge in UCT*. In Proceedings of the 24^{th} International Conference on Machine Learning, pages 273–280, 2007.

[Giordana 00] Attilio Giordana & Lorenza Saitta. *Phase Transitions in Relational Learning*. Machine Learning, vol. 41, no. 2, pages 217–251, 2000.

[Guyon 02] Isabelle Guyon, Jason Weston, Stephen Barnhill & Vladimir Vapnik. *Gene selection for cancer classification using Support Vector Machines*. Machine Learning, vol. 46, no. 1-3, pages 389–422, 2002.

[Guyon 03a] Isabelle Guyon. *Design of experiments for the NIPS 2003 variable selection benchmark*, July 2003.

[Guyon 03b] Isabelle Guyon & André Elisseeff. *An Introduction to Variable and Feature Selection*. Journal of Machine Learning Research, vol. 3, pages 1157–1182, March 2003.

[Guyon 04] Isabelle Guyon, Steve R. Gunn, Asa Ben-Hur & Gideon Dror. *Result Analysis of the NIPS 2003 Feature Selection challenge*. In Proceedings of the 18^{th} Annual Conference on Neural Information Processing Systems, pages 545–552, 2004.

[Guyon 06] Isabelle Guyon, Steve Gunn, Masoud Nikravesh & Lofti Zadeh, editeurs. Feature extraction, foundations and applications. Series Studies in Fuzziness and Soft Computing, Physica-Verlag. Springer, 2006.

[Guyon 07] Isabelle Guyon, Jiwen Li, Theodor Mader, Patrick A. Pletscher, Georg Schneider & Markus Uhr. *Competitive baseline methods set new standards for the NIPS 2003 feature selection benchmark*. Pattern Recognition Letters, vol. 28, no. 12, pages 1438–1444, 2007.

[Guyon 10] Isabelle Guyon, Vincent Lemaire, Marc Boullé, Gideon Dror & David Vogel. *Analysis of the KDD Cup 2009 : Fast Scoring on a Large Orange Customer Database*. In Proceedings of KDD-Cup 2009 competition, volume 7 of *JMLR Workshop and Conference Proceedings*, pages 1–22. JMLR.org, 2010.

[Gärtner 03] Thomas Gärtner, Peter Flach & Stefan Wrobel. *On graph kernels : hardness results and efficient alternatives*. In Proceedings of the 16^{th} annual conference on computational learning theory and the seventh annual workshop on kernel machines, pages 129–143. Springer, 2003.

[Hall 00] Mark A. Hall. *Correlation-based Feature Selection for Discrete and Numeric Class Machine Learning*. In Proceedings of the 17^{th} International Conference on Machine Learning, pages 359–366, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[Hastie 01] Trevor Hastie, Robert Tibshirani & Jerome Friedman. The elements of statistical learning. Springer series in statistics. Springer, New York, 2001.

[Helleputte 09] Thibault Helleputte & Pierre Dupont. *Partially supervised Feature Selection with regularized linear models.* In Proceedings of the 26^{th} International Conference on Machine Learning, pages 409–416, New York, NY, USA, 2009. ACM.

[Hogg 96] Tad Hogg, Bernardo A. Huberman & Colin P. Williams. *Phase Transitions and the search problem.* Artificial Intelligence, vol. 81, no. 1-2, pages 1–15, 1996.

[Hollanders 07] Goele Hollanders, Geert Jan Bex, Marc Gyssens, Ronald L. Westra & Karl Tuyls. *On Phase Transitions in Learning Sparse Networks.* In Proceedings of the 18^{th} European Conference on Machine Learning, pages 591–599, 2007.

[Joachims 98] Thorsten Joachims. *Text Categorization with Suport Vector Machines : Learning with Many Relevant Features.* In Proceedings of the 10^{th} European Conference on Machine Learning, pages 137–142, 1998.

[Joachims 02] Thorsten Joachims. Learning to classify text using Support Vector Machines : Methods, theory and algorithms. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[Karp 83] Richard M. Karp & Judea Pearl. *Searching for an optimal path in a tree with random costs.* Artificial Intelligence, vol. 21, no. 1-2, pages 99–116, 1983.

[Kearns 93] Michael Kearns & Ming Li. *Learning in the presence of malicious errors.* SIAM Journal on Computing, vol. 22, no. 4, pages 807–837, 1993.

[Kira 92] Kenji Kira & Larry A. Rendell. *A practical approach to feature selection.* In Proceedings of the 9^{th} international workshop on Machine learning, pages 249–256, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[Kocsis 06] Levente Kocsis & Csaba Szepesvári. *Bandit based Monte-Carlo planning.* In Proceedings of the 17^{th} European Conference on Machine Learning, pages 282–293. Springer Verlag, 2006.

[Kohavi 97] Ron Kohavi & George H. John. *Wrappers for Feature Subset Selection.* Artificial Intelligence, vol. 97, no. 1-2, pages 273–324, 1997.

[Kononenko 94] Igor Kononenko. *Estimating attributes : analysis and extensions of RELIEF.* In Proceedings of the 7^{th} European Conference on Machine Learning, pages 171–182, Secaucus, NJ, USA, 1994. Springer-Verlag New York, Inc.

[Kramer 00] Stefan Kramer, Nada Lavrač & Peter Flach. *Propositionalization approaches to relational data mining.* Relational Data Mining, pages 262–286, 2000.

[Kwok 07] James T. Kwok & Pak-Ming Cheung. *Marginalized multi-instance kernels.* In Proceedings of the 20^{th} International Joint Conference on Artificial Intelligence, pages 901–906, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[Lai 85] T. L. Lai & Herbert Robbins. *Asymptotically efficient adaptive allocation rules.* Advances in applied mathematics, vol. 6, no. 1, pages 4–22, 1985.

[Lanckriet 04] Gert R. G. Lanckriet, Nello Cristianini, Peter Bartlett, Laurent El Ghaoui & Michael I. Jordan. *Learning the Kernel Matrix with Semidefinite Programming.* Journal of Machine Learning Research, vol. 5, pages 27–72, 2004.

[Landwehr 07] Niels Landwehr, Kristian Kersting & Luc De Raedt. *Integrating Naïve Bayes and FOIL.* Journal of Machine Learning Research, vol. 8, pages 481–507, 2007.

[Langford 09] John Langford, Lihong Li & Tong Zhang. *Sparse Online Learning via Truncated Gradient.* Journal of Machine Learning Research, vol. 10, pages 777–801, 2009.

[Lim 07] Shiau Hong Lim, Li-Lun Wang & Gerald Dejong. *Explanation-Based Feature Construction.* In Proceedings of the 20^{th} International Joint Conference on Artificial Intelligence, 2007.

[Littlestone 89] Nick Littlestone & Manfred K. Warmuth. *The weighted majority algorithm.* In Proceedings of the 30^{th} Annual Symposium on Foundations of Computer Science, pages 256–261, Washington, DC, USA, 1989. IEEE Computer Society.

[Lodhi 02] Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini & Chris Watkins. *Text classification using string kernels.* Journal of Machine Learning Research, vol. 2, pages 419–444, 2002.

[Mahé 05] Pierre Mahé, Nobuhisa Ueda, Tatsuya Akutsu, Jean-Luc Perret & Jean-Philippe Vert. *Graph kernels for molecular structure-activity relationship analysis with Support Vector Machines.* Journal of Chemical Information and Modeling, vol. 45, no. 4, pages 939–51, 2005.

[Mahé 06] Pierre Mahé, Liva Ralaivola, Véronique Stoven & Jean-Philippe Vert. *The Pharmacophore Kernel for Virtual Screening with Support Vector Machines.* Journal of Chemical Information and Modeling, vol. 46, no. 5, pages 2003–2014, 2006.

[Mahé 09] Pierre Mahé & Jean-Philippe Vert. *Graph kernels based on tree patterns for molecules.* Machine Learning, vol. 75, no. 1, pages 3–35, 2009.

[Maloberti 04] Jérôme Maloberti & Michèle Sebag. *Fast Theta-Subsumption with Constraint Satisfaction Algorithms.* Machine Learning, vol. 55, no. 2, pages 137–174, 2004.

[Margaritis 09] Dimitris Margaritis. *Toward Provably Correct Feature Selection in Arbitrary Domains.* In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams & A. Culotta, editeurs, Advances in Neural Information Processing Systems 22, pages 1240–1248, 2009.

[Mézard 02] Marc Mézard & Riccardo Zecchina. *Random K-satisfiability problem : From an analytic solution to an efficient algorithm.* Physical Review E, vol. 66, no. 5, pages 056126–056152, November 2002.

[Mézard 05] Marc Mézard, Thierry Mora & Riccardo Zecchina. *Clustering of solutions in the random satisfiability problem.* Physical Review Letters, vol. 94, no. 19, pages 197205–197208, May 2005.

[Mierswa 05] Ingo Mierswa & Katharina Morik. *Automatic feature extraction for classifying audio data.* Machine Learning, vol. 58, no. 2-3, pages 127–149, 2005.

[Mitchell 92] David Mitchell, Bart Selman & Hector J. Levesque. *Hard and Easy Distributions of SAT Problems.* In Proceedings of the 10^{th} AAAI Spring Symposium, pages 459–465, 1992.

[Monasson 99] Rémi Monasson, Riccardo Zecchina, Scott Kirkpatrick, Bart Selman & Lidror Troyansky. *Determining computational complexity from characteristic 'phase transitions'.* Nature, vol. 400, no. 6740, pages 133–137, 1999.

[Muggleton 91] Stephen Muggleton. *Inductive Logic Programming.* New Generation Computing, vol. 8, no. 4, pages 295–318, 1991.

[Muggleton 95] Stephen Muggleton. *Inverse Entailment and Progol.* New Generation Computing, vol. 13, no. 3&4, pages 245–286, 1995.

[Neal 06] Radford M. Neal & Jianguo Zhang. Feature extraction, foundations and applications, chapitre High Dimensional Classification with Bayesian Neural Networks and Dirichlet Diffusion Trees, pages 265–296. Springer, 2006.

[Pearl 84] Judea Pearl. Heuristics : intelligent search strategies for computer problem solving. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984.

[Pernot 05] Nicolas Pernot, Antoine Cornuéjols & Michèle Sebag. *Phase Transitions within grammatical inference.* In Proceedings of the 19^{th} International Joint Conference on Artificial Intelligence, pages 811–816, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc.

[Plotkin 70] Gordon D. Plotkin. *A Note on Inductive Generalization.* Machine Intelligence, vol. 5, pages 153–163, 1970.

[Purdom 83] Paul Walton Purdom Jr. *Search rearrangement backtracking and polynomial average time.* Artificial Intelligence, vol. 21, no. 1-2, pages 117–133, 1983.

[Quinlan 90] J. Ross Quinlan. *Learning Logical Definitions from Relations.* Machine Learning, vol. 5, no. 3, pages 239–266, 1990.

[Rakotomamonjy 08] Alain Rakotomamonjy, Francis Bach, Stephane Canu & Yves Grandvalet. *SimpleMKL.* Journal of Machine Learning Research, vol. 9, pages 2491–2521, 2008.

[Rimmel 10] Arpad Rimmel, Fabien Teytaud & Olivier Teytaud. *Biasing Monte-Carlo Simulations through RAVE Values.* In Proceedings of the International Conference on Computers and Games, Kanazawa, Japon, September 2010.

[Rissanen 78] Jorma Rissanen. *Modeling by shortest data description.* Automatica, vol. 14, no. 5, pages 465–471, September 1978.

[Robbins 52] Herbert Robbins. *Some aspects of the sequential design of experiments.* Bulletin of the American Mathematical Society, vol. 58, no. 5, pages 527–535, 1952.

[Robnik-Šikonja 03] Marko Robnik-Šikonja & Igor Kononenko. *Theoretical and Empirical Analysis of ReliefF and RReliefF.* Machine Learning, vol. 53, no. 1-2, pages 23–69, 2003.

[Rogers 05] Jeremy Rogers & Steve R. Gunn. *Identifying Feature Relevance Using a Random Forest.* In Craig Saunders, Marko Grobelnik, Steve R. Gunn & John Shawe-Taylor, editeurs, In Proceedings of the Subspace, Latent Structure and Feature Selection techniques : Statistical and Optimization, Perspectives Workshop, Revised Selected Papers, volume 3940 of *Lecture Notes in Computer Science*, pages 173–184. Springer, February 2005.

[Rolet 09] Philippe Rolet, Michèle Sebag & O. Teytaud. *Boosting Active Learning to optimality : a tractable Monte-Carlo, Billiard-based algorithm.* In Proceedings of the 20^{th} European Conference on Machine Learning, pages 302–317. Springer Verlag, 2009.

[Rückert 02] Ulrich Rückert, Stefan Kramer & Luc De Raedt. *Phase Transitions and Stochastic Local Search in k-Term DNF Learning.* In Proceedings of the 13^{th} European Conference on Machine Learning, pages 405–417, 2002.

[Rückert 03] Ulrich Rückert & Stefan Kramer. *Stochastic Local Search in k-Term DNF Learning.* In Proceedings of the 20^{th} International Conference on Machine Learning, pages 648–655, 2003.

[Saitta 10] Lorenza Saitta, Attilio Giordana & Antoine Cornuéjols. Phase Transitions in Machine Learning. Cambridge University Press, 2010.

[Schölkopf 95] Bernhard Schölkopf, Chris Burges & Vladimir Vapnik. *Extracting Support Data for a Given Task.* In Proceedings of the First International Conference on Knowledge Discovery and Data Mining, pages 252–257. AAAI Press, 1995.

[Schölkopf 04] Bernhard Schölkopf, Koji Tsuda & Jean-Philippe Vert. Kernel methods in computational biology. MIT Press, 2004.

[Selman 94] Bart Selman, Henry A. Kautz & Bram Cohen. *Noise strategies for improving local search.* In Proceedings of the 12^{th} AAAI National Conference On Artificial Intelligence, pages 337–343, Menlo Park, CA, USA, 1994. American Association for Artificial Intelligence.

[Shen 08] Kai Quan Shen, Chong Jin Ong, Xiao Ping Li & Einar P. V. Wilder-Smith. *Feature selection via sensitivity analysis of SVM probabilistic outputs.* Machine Learning, vol. 70, no. 1, pages 1–20, 2008.

[Sonnenburg 06] Sören Sonnenburg, Gunnar Rätsch, Christin Schäfer & Bernhard Schölkopf. *Large Scale Multiple Kernel Learning.* Journal of Machine Learning Research, vol. 7, pages 1531–1565, 2006.

[Srinivasan 94] Ashwin Srinivasan, Stephen Muggleton, Michael J. E. Sternberg & Ross D. King. *Mutagenesis : ILP experiments in a non-determinate biological domain.* In Proceedings of the 4^{th} Inductive Logic Programming Workshop, 1994.

[Stoppiglia 03] Hervé Stoppiglia, Gérard Dreyfus, Rémi Dubois & Yacine Oussar. *Ranking a random feature for variable and feature selection.* Journal of Machine Learning Research, vol. 3, pages 1399–1414, 2003.

[Sutton 98] Richard S. Sutton & Andrew G. Barto. Reinforcement learning : An introduction. MIT Press (Bradford Book), 1998.

[Teytaud 07] Olivier Teytaud, Sylvain Gelly & Michèle Sebag. *Anytime Many-Armed Bandits.* In Compte-rendu de la Conférence francophone en Apprentissage, Grenoble, France, 2007.

[Teytaud 09] Fabien Teytaud & Olivier Teytaud. *Creating an Upper-Confidence-Tree program for Havannah.* In Proceedings of the 12^{th} conference on Advances in Computer Games, Pamplona, Spain, 2009.

[Tibshirani 94] Robert Tibshirani. *Regression shrinkage and selection via the Lasso.* Journal of the Royal Statistical Society, Series B, vol. 58, pages 267–288, 1994.

[Torkkola 03] Kari Torkkola. *Feature extraction by non parametric mutual information maximization.* Journal of Machine Learning Research, vol. 3, pages 1415–1438, 2003.

[Tuv 09] Eugene Tuv, Alexander Borisov, George Runger & Kari Torkkola. *Feature Selection with Ensembles, Artificial Variables, and Redundancy Elimination.* Journal of Machine Learning Research, vol. 10, pages 1341–1366, 2009.

[Vapnik 98] Vladimir Naumovich Vapnik. Statistical learning theory. Wiley, New-York, 1998.

[Varma 09] Manik Varma & Bodla Rakesh Babu. *More generality in efficient Multiple Kernel Learning.* In Proceedings of the 26^{th} International Conference on Machine Learning, pages 1065–1072, New York, NY, USA, 2009. ACM.

[Vilalta 02] Ricardo Vilalta & Youssef Drissi. *A perspective view and survey of meta-learning.* Artificial Intelligencnce Review, vol. 18, no. 2, pages 77–95, 2002.

[Wang 08] Yizao Wang, Jean Yves Audibert & Rémi Munos. *Algorithms for Infinitely Many-Armed Bandits.* In Proceedings of the 22^{nd} Annual Conference on Neural Information Processing Systems, pages 1729–1736, 2008.

[Wei 04] Wei Wei, Jordan Erenrich & Bart Selman. *Towards Efficient Sampling : Exploiting Random Walk Strategies.* In Proceedings of the 19^{th} AAAI National Conference On Artificial Intelligence, pages 670–676, 2004.

[Weidmann 03] Nils Weidmann, Eibe Frank & Bernhard Pfahringer. *A two-level learning method for generalized Multi-Instance problems.* In Proceedings of the 14^{th} European Conference on Machine Learning, 2003.

[Weston 03] Jason Weston, André Elisseeff, Bernhard Schölkopf & Mike Tipping. *Use of the zero norm with linear models and kernel methods.* Journal of Machine Learning Research, vol. 3, pages 1439–1461, 2003.

[Xu 05] Ke Xu, Frédéric Boussemart, Fred Hemery & Christophe Lecoutre. *A simple model to generate hard satisfiable instances.* In Proceedings of the 19^{th} International Joint Conference on Artificial Intelligence, pages 337–342, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc.

[Xu 09] Zenglin Xu, Rong Jin, Jieping Ye, Michael R. Lyu & Irwin King. *Non-monotonic Feature Selection.* In Proceedings of the 26^{th} International Conference on Machine Learning, pages 1145–1152, New York, NY, USA, 2009. ACM.

[Zhang 04] Weixiong Zhang. *Phase transitions and backbones of the asymmetric traveling salesman problem.* Journal of Artificial Intelligence Research, vol. 21, no. 1, pages 471–497, 2004.

[Zhang 08a] Tong Zhang. *Adaptive Forward-Backward Greedy Algorithm for Sparse Learning with Linear Models.* In Proceedings of the 22^{nd} Annual Conference on Neural Information Processing Systems, pages 1921–1928, 2008.

[Zhang 08b] Tong Zhang. *Multi-stage Convex Relaxation for Learning with Sparse Regularization.* In Proceedings of the 22^{nd} Annual Conference on Neural Information Processing Systems, pages 1929–1936, 2008.

[Zien 07] Alexander Zien & Cheng Soon Ong. *Multiclass Multiple Kernel Learning.* In Proceedings of the 24^{th} International Conference on Machine Learning, pages 1191–1198, New York, NY, USA, 2007. ACM.