

Antoine Cornuéjols - Laurent Miclet

Avec la participation d'Yves Kodratoff

Apprentissage artificiel

Concepts et algorithmes

Préface de Tom Mitchell

EYROLLES

Apprentissage artificiel

Antoine Cornuéjols est maître de conférences à l’Institut d’informatique d’entreprise et chercheur au LRI de Paris XI à Orsay. Il enseigne l’apprentissage artificiel dans plusieurs grandes écoles et en DEA. Ses recherches portent notamment sur l’utilisation de l’apprentissage pour l’analyse de données en médecine, en génomique et en vision artificielle.

Laurent Miclet est professeur à l’ENSSAT de Lannion. Il est responsable du projet CORDIAL de l’INRIA et enseigne l’apprentissage artificiel et la reconnaissance des formes dans plusieurs grandes écoles et en DEA. Ses recherches portent en particulier sur l’apprentissage pour le dialogue homme-machine et les technologies vocales.

Yves Kodratoff est directeur de recherches au CNRS et dirige au LRI l’équipe Inférence et Apprentissage. Il s’intéresse à toutes les techniques de raisonnement inductif, et en particulier à leur application au data mining.

Cet ouvrage est publié avec le concours de l’École Nationale Supérieure des Sciences Appliquées et de Technologie (Lannion).

- Les programmes d’intelligence artificielle sont aujourd’hui capables de reconnaître des commandes vocales, d’analyser automatiquement des photos satellites, d’assister des experts pour prendre des décisions dans des environnements complexes et évolutifs (analyse de marchés financiers, diagnostics médicaux...), de fouiller d’immenses bases de données hétérogènes, telles les innombrables pages du Web...
- Pour réaliser ces tâches, ils sont dotés de modules d’apprentissage leur permettant d’adapter leur comportement à des situations jamais rencontrées, ou d’extraire des lois à partir de bases de données d’exemples.
- Ce livre présente les concepts qui sous-tendent l’apprentissage artificiel, les algorithmes qui en découlent et certaines de leurs applications. Son objectif est de décrire un ensemble d’algorithmes utiles en tentant d’établir un cadre théorique unique pour l’ensemble des techniques regroupées sous ce terme « d’apprentissage artificiel ».
- À qui s’adresse ce livre ?
 - Aux décideurs et aux ingénieurs qui souhaitent comprendre l’apprentissage automatique et en acquérir des connaissances solides ;
 - Aux étudiants de niveau maîtrise, DEA ou école d’ingénieurs qui souhaitent un ouvrage de référence en intelligence artificielle et en reconnaissance des formes.

► Sommaire

- I. Les fondements de l’apprentissage • Première approche théorique de l’induction • Environnement méthodologique • II. Apprentissage par exploration • Induction et relation d’ordre • Programmation logique inductive • Inférence grammaticale • Apprentissage par évolution • III. Apprentissage par optimisation • Surfaces séparatrices linéaires • Réseaux connexionnistes • Réseaux bayésiens • Modèles de Markov cachés • IV. Apprentissage par approximation et interpolation • Classification non supervisée • Apprentissage par renforcement • Annexes et bibliographie.

► Collection Technique et Scientifique des Télécommunications publiée sous l’égide de France Telecom Recherche et Développement

EYROLLES

Apprentissage artificiel

Concepts et algorithmes

DANS LA MÊME COLLECTION

G. DREYFUS et al. – Réseaux de neurones.

Méthodologie et applications.

N°11019, 2002, 380 pages.

Y. COLLETTE, P. SIARRY. – Optimisation multiobjectif.

N°11168, 2002, 328 pages.

C. GUÉRET, C. PRINS, M. SEVAUX. – Programmation linéaire.

65 problèmes d'optimisation modélisés et résolus avec Visual XPress.

N°9202, 2^e édition, mars 2003, 365 pages + CD-Rom.

CHEZ LE MÊME ÉDITEUR

R. LEFÉBURE, G. VENTURI. – Data mining.

Gestion de la relation client –Personnalisation de sites Web.

N°9176, 2001, 392 pages, avec CD-ROM.

M. BAZSALICZA, P. NAÏM. – Data mining pour le Web.

Profiling –filtrage collaboratif – Personnalisation client.

N°9203, 2001, 280 pages.

M. JAMBU. – Méthodes de base de l'analyse des données.

N°5256, 1999, 440 pages.

M. GONDTRAN, M. MINOUX. – Graphes et algorithmes.

N°1571, 1995, 622 pages.

BOURDA. – Introduction à l'informatique théorique.

N°1642, 1994, 236 pages.

Apprentissage artificiel

Concepts et algorithmes

Antoine Cornuéjols • Laurent Miclet

Avec la participation d'Yves Kodratoff

Deuxième tirage 2003

EYROLLES



ÉDITIONS EYROLLES
61, Bld Saint-Germain
75240 Paris cedex 05
www.editions-eyrolles.com



Le code de la propriété intellectuelle du 1^{er} juillet 1992 interdit en effet expressément la photocopie à usage collectif sans autorisation des ayants droit. Or, cette pratique s'est généralisée notamment dans les établissements d'enseignement, provoquant une baisse brutale des achats de livres, au point que la possibilité même pour les auteurs de créer des œuvres nouvelles et de les faire éditer correctement est aujourd'hui menacée.

En application de la loi du 11 mars 1957, il est interdit de reproduire intégralement ou partiellement le présent ouvrage, sur quelque support que ce soit, sans autorisation de l'Éditeur ou du Centre Français d'Exploitation du Droit de Copie, 20, rue des Grands-Augustins, 75006 Paris.

© Groupe Eyrolles, 2002, ISBN : 2-212-11020-0

© France Telecom Recherche et Développement, 2002, ISBN : 2-212-11162-2

Préface

LAPPRENTISSAGE ARTIFICIEL s'intéresse à l'écriture de programmes d'ordinateur capables de s'améliorer automatiquement au fil du temps, soit sur la base de leur propre expérience, soit à partir de données antérieures fournies par d'autres programmes. Dans le domaine scientifique relativement jeune de l'informatique, l'apprentissage artificiel joue un rôle de plus en plus essentiel. Au début de son existence, dans les années 1950, l'informatique se résumait principalement à programmer des machines en leur spécifiant ligne après ligne la séquence d'instructions que l'ordinateur aurait à suivre. Autour des années 1990, les logiciels étaient devenus si complexes qu'une alternative s'imposait naturellement : développer des techniques pour que les programmes puissent s'entraîner à partir d'exemples. Le résultat est qu'il existe aujourd'hui de nombreux domaines d'application de l'informatique dans lesquels les méthodes de l'apprentissage artificiel sont employées pour entraîner les logiciels. Mieux, le code résultant dépasse de beaucoup en performance les réalisations les plus abouties de programmation manuelle « ligne après ligne ». C'est ainsi que tous les meilleurs logiciels commercialisés de reconnaissance de la parole sont fondés sur l'entraînement de leurs programmes à la reconnaissance des différents sons et mots. La plupart d'entre eux permettent même à l'utilisateur d'accoutumer le système aux caractéristiques de sa voix. D'autres exemples existent dans des domaines tels que la vision par ordinateur, le traitement automatique du texte et la robotique.

La discipline de l'apprentissage artificiel peut donc déjà revendiquer des succès dans un grand nombre de domaines d'application. Des logiciels de fouille de données sont utilisés à grande échelle pour découvrir quelle prescription est la plus efficace pour quel patient, à partir de l'analyse de fichiers médicaux antérieurs. La palette des applications va de la prédiction de la demande en énergie, étant connu l'historique des consommations antérieures, à l'apprentissage de la reconnaissance de transactions frauduleuses par carte de crédit, par examen des transactions passées avérées frauduleuses. Au moment où nous passons des cinquante premières années de l'informatique au les cinquante prochaines, il semble certain que le rôle de l'apprentissage artificiel ne cessera de croître au centre de cette science.

Pourquoi cette progression ? La réponse fondamentale est que nous possédons désormais la compréhension de plusieurs principes calculatoires qui guident tout processus d'apprentissage, qu'il soit implanté sur une machine ou sur un humain. La discipline de l'apprentissage artificiel possède désormais de riches fondements théoriques : on commence à savoir répondre à des questions comme : « Combien au minimum d'exemples d'entraînement faut-il fournir à un programme d'apprentissage pour être certain qu'il apprenne avec une efficacité donnée ? » et « Quelles méthodes d'apprentissage sont les plus efficaces pour tel ou tel type de problème ? » Ces fondements proviennent de la théorie statistique de l'estimation, de la théorie de l'identification et de la commande optimale, de travaux pionniers sur la complexité de l'apprentissage de grammaires ou plus récents sur l'inférence bayésienne algorithmique.

Cet ouvrage fournit au lecteur francophone l'introduction la plus complète à ce jour à l'apprentissage artificiel. Il traite de la théorie et des applications de cette discipline sous un grand nombre d'aspects, en couvrant des sujets comme l'apprentissage bayésien, l'inférence grammaticale ou l'apprentissage par renforcement. C'est avec plaisir que je recommande au lecteur de découvrir ce livre, et à travers lui les idées et les méthodes de l'apprentissage artificiel.

Tom M. MITCHELL

Pittsburgh, Pennsylvania, USA
Le 29 Mai 2002

*The idea of a learning machine may appear paradoxical to some readers.
A. M. Turing, 1950.*

*à Isabelle, Claire, Aurélie, Sébastien, Fanny
et à Maura, Fabien, Marion*

Remerciements

Ce livre est publié avec l'aide de l'ENSSAT. Nous remercions son directeur, Joël Crestel, d'avoir associé cet établissement à la publication de cet ouvrage.

Nous devons des remerciements particuliers aux personnes qui nous ont autorisés à reprendre leurs écrits, édités ou non, ainsi qu'à toutes celles qui nous ont fait bénéficier de leur expertise pour nous aider dans la rédaction de cet ouvrage. Notre gratitude va aussi aux lecteurs critiques des versions préliminaires, ce qui inclut notamment une certaine proportion de nos étudiants.

Il nous tient à cœur de remercier tout spécialement : Abdel Belaïd, Sami Bengio, Christophe Bernard, Marc Bernard, Olivier Boëffard, Michel Cartier, Christophe Choisy, François Coste, François Denis, Pierre Dupont, Daniel Fredouille, Colin de la Higuera, Yves Kodratoff, Israël-César Lerman, Stan Matwin, Engelbert Mephu Nguifo, Tom Mitchell, Jacques Nicolas, Céline Rouveiro, Michèle Sebag, Dominique Snyers, Franck Thollard, Fabien Torre, Stéphane Vandemersch et Jean-Daniel Zucker.

L'adresse Web : www.editions-eyrolles.com contient les figures de cet ouvrage, les transparents des cours des auteurs et les *errata*.

Avant-propos

CE LIVRE présente les théories, les algorithmes et les applications de l'apprentissage artificiel. Son ambition est d'une part d'unifier le cadre méthodologique, et d'autre part de décrire un ensemble d'algorithmes utiles, de manière cohérente avec ce cadre, enfin de faire connaître ses applications existantes et potentielles.

À quoi sert l'apprentissage artificiel ? La plupart des programmes d'intelligence artificielle possèdent aujourd'hui un module d'apprentissage et tous les programmes de reconnaissance des formes sont fondés sur des algorithmes d'apprentissage. Et que font ces programmes ? Ils sont capables de reconnaître la parole humaine et de l'interpréter. Ils réalisent une analyse automatique de photos satellites pour détecter certaines ressources sur la Terre. Ils assistent les experts pour prendre des décisions dans des environnements complexes et évolutifs, par exemple le marché financier ou le diagnostic médical. Ils fouillent d'immenses bases de données hétérogènes comme les millions de pages Web accessibles à tous. Ils analysent les données clientèle des entreprises pour les aider à mieux cibler leurs campagnes de publicité. Ils participent aussi à des tournois : le 11 mai 1997, le tenant du titre de champion du monde du jeu d'échecs, Gary Kasparov, a été battu en match par un programme.

On sait donc programmer les ordinateurs pour leur faire exécuter des tâches considérées comme intelligentes, de multiples façons et de manière de plus en plus efficace. Cet ouvrage s'intéresse à un aspect particulier de cette intelligence artificielle : la faculté d'apprentissage.

L'apprentissage artificiel est une discipline dont les outils et les champs d'applications sont assez disparates. Cependant, les connaissances de base nécessaires à sa compréhension sont essentiellement une culture généraliste que l'on trouve par exemple dans les ouvrages de mathématiques pour l'informatique : notions d'algèbre linéaire, de probabilités, de combinatoire, d'analyse élémentaire, d'algorithmique, de théorie des langages, de logique. Dans la mesure du possible, ces notions de base sont brièvement rappelées selon la nécessité des chapitres de ce livre.

À qui s'adresse cet ouvrage ?

On peut tirer profit de cet ouvrage en autodidacte, comme le fera par exemple un ingénieur qui cherche à connaître ce qui se cache derrière les mots ou à acquérir une initiation à des techniques qu'il ignore encore. On pourra aussi s'en servir comme d'un appui pour compléter un enseignement : ce sera le cas pour un étudiant au niveau maîtrise, DEA ou école d'ingénieurs, ou comme d'un ouvrage de référence pour faire un cours sur le domaine.

Quelques applications de l'apprentissage artificiel

Voyons maintenant comment rendre un programme plus efficace en le dotant d'une possibilité d'apprentissage. Reprenons pour cela les applications de l'intelligence artificielle et de la reconnaissance des formes citées ci-dessus.

- Un programme de reconnaissance de la parole augmente ses performances au fur et à mesure de son utilisation par la même personne: c'est une expérience qu'il est aujourd'hui facile de faire en pratique si on achète un logiciel personnel de dictée vocale.
- Un programme de détection des ressources terrestres apprend à reconnaître une zone de pollution au milieu de la mer, à partir d'une base de données d'exemples d'images de zones connues comme propres ou comme polluées: cette base de données lui sert d'expérience pour déterminer sa décision sur une zone inconnue.
- Un programme de diagnostic sur un ensemble d'informations évolutives prises sur un patient doit avoir été pourvu de connaissances, à partir de diagnostics de praticiens et d'experts sur des situations types. Mais il doit aussi avoir été doté d'un module de généralisation, de façon à réagir correctement à des situations auxquelles il n'a jamais été confronté exactement.
- Les moteurs de recherche sur le Web pourraient être munis d'un module d'adaptation au style de navigation de l'usager : c'est une faculté souhaitable pour augmenter l'ergonomie de leur utilisation. Les programmes ne sont pas encore réellement agrémentés de cette propriété, mais il est clair que c'est une condition nécessaire pour franchir certains obstacles de communication si évidents actuellement.
- L'exploitation des fichiers client d'une entreprise est souvent faite par un expert ou un programme expert qui utilise des règles explicites pour cibler un segment de clientèle susceptible d'être intéressé par un nouveau produit. Mais ces règles peuvent être acquises automatiquement, par un apprentissage dont le but est de fournir de nouvelles connaissances expertes, à la fois efficaces et intelligibles pour l'expert.
- Un programme de jeu d'échecs possède en général une très bonne efficacité *a priori*; mais il est naturel d'essayer de le doter d'un module où il puisse analyser ses défaites et ses victoires, pour améliorer ses performances moyennes dans ses parties futures. Ce module d'apprentissage existe dans un certain nombre de programmes de jeux.

Quelques définitions de base

Apprentissage (sous entendu : artificiel, automatique) (*Machine Learning*)

Cette notion englobe toute méthode permettant de construire un modèle de la réalité à partir de données, soit en améliorant un modèle partiel ou moins général, soit en créant complètement le modèle. Il existe deux tendances principales en apprentissage, celle issue de l'intelligence artificielle et qualifiée de *symbolique*, et celle issue des statistiques et qualifiée de *numérique*.

Fouille de données (*Data Mining*) ou Extraction de connaissances à partir des données (*Knowledge Discovery in Data*)

La fouille de données prend en charge le processus complet d'extraction de connaissances : stockage dans une base de données, sélection des données à étudier, si nécessaire : nettoyage des données, puis utilisation des apprentissages numériques et symboliques afin de proposer des modèles à l'utilisateur, enfin validation des modèles proposés. Si ces modèles sont invalidés par l'utilisateur, le processus complet est répété.

Précision vs. Généralisation

Le grand dilemme de l'apprentissage. La précision est définie par un écart entre une valeur mesurée ou prédite et une valeur réelle. Apprendre avec trop de précision conduit à un « sur-apprentissage », comme l'apprentissage par cœur, pour lequel des détails insignifiants (ou dûs au bruit) sont appris. Apprendre avec trop peu de précision conduit à une « sur-généralisation » telle que le modèle s'applique même quand l'utilisateur ne le désire pas. Les deux types d'apprentissage, numérique et symbolique, ont défini des mesures de généralisation et c'est à l'utilisateur de fixer le seuil de généralisation qu'il juge optimal.

Intelligibilité (devrait être *Comprehensibility* mais tend à devenir *Understandability*)

Depuis quelques années, principalement sous la poussée des industriels, les chercheurs se sont mis à essayer de contrôler aussi l'intelligibilité du modèle obtenu par la fouille de données. Jusqu'à présent les méthodes de mesure de l'intelligibilité se réduisent à vérifier que les résultats sont exprimés dans le langage de l'utilisateur et que la taille des modèles n'est pas excessive. Des méthodes spécifiques de visualisation sont aussi utilisées.

Classification, classement et régression.

La classification, telle qu'elle est définie en analyse de données, consiste à regrouper des ensembles d'exemples non supervisés en classes. Ces classes sont souvent organisées en une structure (*clustering*). Si cette structure est un arbre, alors on parle de taxonomie ou de taxinomie (*taxonomy*). Sous l'influence du mot anglais *classification*, on a tendance à confondre classification et classement. Ce dernier mot désigne le processus de reconnaissance en intension (par leur propriétés) de classes décrites en extension (par les valeurs de leurs descripteurs). Lorsque les valeurs à prédire sont des classes en petit nombre, on parle de classification. Il s'agit par exemple de prévoir l'appartenance d'un oiseau observé à la classe « canard » ou « oie ». La *régression* traite des cas où les valeurs à prédire sont numériques, par exemple: *nombre d'exemplaires de cet ouvrage qui seront vendus = 3900*.

Deux champs industriels de l'apprentissage artificiel : la reconnaissance des formes et la fouille de données

En quarante ans et plus d'existence, l'apprentissage artificiel a fourni un grand nombre d'outils aux industriels et aux entrepreneurs. Nous les regroupons selon deux grands axes : la *reconnaissance des formes* et la *fouille de données* ou pour être plus précis, l'*extraction de connaissances des données*.

Le second domaine est le moins bien connu des deux bien qu'il soit porteur de réelles possibilités économiques.

Quant au premier, rappelons seulement que les méthodes de l'apprentissage artificiel sont à la base de la reconnaissance des images (écriture manuscrite, signatures, détection de ressources par satellite, pilotage automatique, etc.), de la reconnaissance de la parole, du traitement avancé des signaux bio-médicaux, etc. Pour mesurer l'extraordinaire vitalité des applications et des potentialités de la reconnaissance des formes, il suffit par exemple de suivre la parution incessante des livres dans ce domaine. Pour ne citer que lui, l'éditeur World Scientific a une cinquantaine de titres à son catalogue sous la rubrique « Applications de la reconnaissance des formes » et les renouvelle à raison de près d'une dizaine par an.

Revenons maintenant à la fouille de données. Les problèmes pratiques que peut résoudre en ce domaine l'apprentissage artificiel se posent constamment dans la vie industrielle: comment distinguer un bon client d'un mauvais, comment reconnaître un mauvais procédé de fabrication et l'améliorer, voilà deux exemples frappants parmi d'autres. On constate pourtant que l'ancrage

de ce type d'application dans la vie industrielle ne date que des années 1990, avec la naissance d'un discipline nouvelle, créée sous le nom de « fouille de données » (*data mining*) ou ECD : « extraction de connaissances à partir des données » (*knowledge discovery in databases*, KDD). Nous présentons rapidement le domaine avant d'en donner l'état de l'art industriel dans le dernier paragraphe de cet avant-propos.

L'ECD est née de la constatation que les trois approches qui permettaient de construire des modèles, à savoir les statistiques exploratoires, l'analyse des données et l'apprentissage symbolique automatique (ASA), souffraient de deux défauts communs : exiger des données présentées sous une forme très rigide et faire peu de cas de l'intelligibilité des résultats. De plus, chacune présentait un défaut particulier gênant leur emploi : les statistiques exploratoires et l'analyse des données s'adressaient à des données essentiellement numériques et l'ASA se limitait aux données symboliques ou discrétisées en intervalles de valeurs.

Depuis, ces domaines ont évolué et les critiques à leur adresser ont changé, mais tel était l'état de l'art dans les années 1990. L'ECD est donc née d'un quadruple effort :

- permettre aux utilisateurs de fournir des données dans l'état où elles sont : ceci a donné naissance aux techniques de nettoyage des données (ce point sera développé au chapitre 3) ;
- utiliser les données enregistrées sous forme de bases de données (en général relationnelles) : ceci a provoqué un large courant de recherche au sein de la communauté des BD intéressée par la création de modèles ;
- fournir aux utilisateurs des outils capables de travailler sur des données mixtes, numériques et symboliques ;
- construire des outils produisant une connaissance intelligible aux utilisateurs.

C'est ainsi que l'ECD a pu trouver la large reconnaissance industrielle dont elle jouit actuellement. Elle a commencé à résoudre les deux problèmes industriels principaux de l'analyse des données, ceux qui coûtent le plus cher : le fait que le client est souvent imprécis dans la définition du problème qu'il se pose et le fait que les données dont il dispose sont souvent de qualité discutable.

L'étude des applications industrielles de l'ECD montre qu'il existe une assez forte demande en outils de création de modèles, autrement dit en apprentissage artificiel. Ceci se traduit par le fait qu'environ cent cinquante compagnies se sont spécialisées dans ce domaine. Certaines de ces compagnies existent depuis plusieurs années et d'autres se sont vendues fort cher. L'ensemble revèle bien un secteur en progression raisonnable sur plusieurs années.

Notre estimation est que le marché de l'ECD est occupé par 60 % d'outils d'apprentissage statistique et 40 % d'outils d'apprentissage symboliques. Ces dernières techniques étant moins enseignées que les premières dans les universités, on constate un hiatus entre l'enseignement et l'industrie. En tous cas, le présent livre cherche à aller dans le sens d'un meilleur enseignement des méthodes de l'apprentissage artificiel, symbolique comme statistique.

Les caractéristiques de l'apprentissage artificiel

Certaines des facultés que l'on peut lier naturellement à l'apprentissage ont été citées dans les exemples ci-dessus : entraînement, reconnaissance, généralisation, adaptation, amélioration, intelligibilité.

Rappelons la définition classique de l'apprentissage en sciences cognitives : « capacité à améliorer les performances au fur et à mesure de l'exercice d'une activité ». Cette définition s'applique en particulier au comportement d'un joueur d'échecs au fil des parties, où l'assimilation de l'expérience et la puissance de raisonnement se combinent dans sa progression. Elle

est aussi pertinente pour des tâches perceptives : on s'habitue à un accent, à une écriture. On accumule des bonnes et des mauvaises expériences. À partir d'elles, on sait, consciemment ou non, en abstraire ou faire évoluer des règles pour mieux effectuer la tâche.

Nous avons mentionné une autre facette de l'apprentissage, souvent entremêlée à la précédente : la faculté à généraliser rationnellement. Si une expérience accumulée sur un certain nombre d'exemples a permis de tirer des règles de comportement, ces règles doivent s'appliquer aussi à des situations non encore rencontrées. Prenons quelqu'un qui apprend à conduire sur une berline de petite puissance. Dès qu'il a mérité le permis, la loi l'autorise à conduire une camionnette utilitaire ou une voiture de sport. C'est que les règles qu'il a apprises et les réflexes qu'il a acquis s'appliquent aussi (plus ou moins directement) à ces véhicules.

Qu'en est-il des machines ? Dès les débuts de l'intelligence artificielle, c'est-à-dire en vérité dès l'apparition des ordinateurs, les chercheurs et les ingénieurs se sont posés le problème de l'apprentissage¹. L'apprentissage artificiel dans sa situation actuelle est donc le produit d'une histoire de cinquante ans de recherches et de réalisations. Comme on l'a vu, un grand nombre de tâches d'intelligence artificielle et de reconnaissance des formes s'appuient ou sont fondées sur des modules d'apprentissage.

On verra dans cet ouvrage comment des programmes peuvent mettre en œuvre un apprentissage par amélioration du comportement, en général grâce à des techniques d'optimisation. On verra aussi qu'il est possible d'écrire des programmes qui réalisent un apprentissage par généralisation : quand on leur donne suffisamment d'exemples et le type du concept à apprendre, ils choisissent un concept qui n'est pas seulement valide sur les exemples qu'ils ont vus, mais qui sera également valable pour d'autres. C'est ainsi qu'un programme de reconnaissance de la parole ne peut pas « entendre » tous les sons avant d'élaborer une règle de décision. Il est écrit pour extraire une méthode de classification de ceux qu'on lui a présentés et traiter ensuite du mieux possible tous les sons qu'il aura à décoder.

En réalité, d'un point de vue informatique, la problématique n'est pas fondamentalement différente dans les deux cas. Il s'agit dans le premier de faire évoluer des règles de comportement au fil des exemples et dans le second d'extraire des règles à partir d'un ensemble d'exemples donné *a priori*. De même que dans l'apprentissage naturel, un panachage de ces deux modes de fonctionnement est facile à concevoir dans l'apprentissage artificiel.

Il y a une autre facette de l'apprentissage que l'intelligence artificielle explore. Quand un expert extrait des connaissances d'un ensemble de données, il apprend une certaine façon de les résumer. Mais le résultat de cet apprentissage ne sera opératoire que si la connaissance extraite est intelligible, transmissible de l'expert aux utilisateurs, interprétable « en clair ». Il en est de même pour un agent artificiel : certaines tâches d'apprentissage ne se mesurent pas seulement par leur qualité de prédiction, mais aussi par la manière dont les résultats sont expliqués. Cet aspect est relié opérationnellement à l'intelligence artificielle symbolique, aux systèmes experts en particulier : mieux vaut souvent un petit nombre de règles compréhensibles qu'un fouillis de règles sophistiquées, même avec une performance objective supérieure.

Avant de décrire plus en détail les motivations et l'organisation de cet ouvrage, précisons à travers trois exemples comment s'organise l'apprentissage dans des situations concrètes. Cela nous permettra de donner une typologie des méthodes et de présenter le plan de cet ouvrage.

1. A. Turing, dans son article *Computing Machine and Intelligence*, de la revue *Mind* en Octobre 1950 (Vol LIX, No 236) avait intitulé un paragraphe *Learning Machines*. On peut consulter un fac-similé du manuscrit sur le site <http://data.archives.ecs.soton.ac.uk/turing/> et le texte à : <http://www.abelard.org/turpap/turpap.htm>

Trois exemples d'apprentissage

Un exemple ornithologique

Imaginons un étang sur lequel nagent des oies et des cygnes (nous admettons qu'il n'y a pas d'autres oiseaux dans cette région). Le brouillard est tombé, quand arrivent deux avimateurs dont l'un est expert et l'autre débutant. Ils n'aperçoivent en arrivant qu'une partie des animaux, de manière peu distincte. Pour l'expert, l'identification est cependant facile (il n'est pas expert pour rien). Quant au débutant, il doit se contenter de mesurer ce qui lui paraît caractéristique : le niveau de gris du plumage et la taille de la bête. Pour représenter le problème, il va donc prendre ces deux mesures sur chaque animal qu'il voit et faire un graphique : il se place ainsi dans un certain espace de représentation (figure 0.1, à gauche).

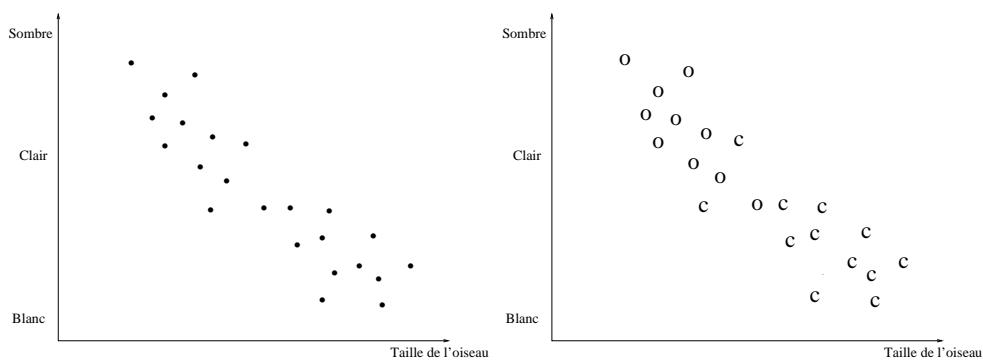


FIG. 0.1 – *Le premier graphique de l'avimateur débutant représente les oiseaux observés placés dans son espace de représentation. Le second graphique représente les mêmes oiseaux, mais il est étiqueté par l'expert. La lettre O signifie que l'oiseau est une oie, C qu'il est un cygne.*

Maintenant, comment lancer une phase d'apprentissage ? Il faut que le débutant se place en situation d'apprenant vis-à-vis de l'expert, en lui demandant quelle est la décision correcte pour chaque oiseau. Une fois que l'expert a agi comme un professeur en donnant toutes les réponses, notre apprenant possède un graphique enrichi (figure 0.1, à droite) qui va lui permettre de démarrer l'apprentissage proprement dit.

Le problème d'apprentissage est maintenant bien posé. Il peut s'énoncer ainsi : comment trouver une règle qui décide, dans l'espace de représentation choisi, avec le moins d'erreurs possibles, quel oiseau est une oie et quel oiseau est un cygne ? La règle trouvée doit posséder de bonnes propriétés de généralisation, c'est-à-dire fonctionner au mieux non seulement sur ces exemples expertisés, mais par la suite sur des oiseaux non encore observés.

Que sera une telle règle ? L'apprenant peut imaginer de tracer dans le plan de représentation une ligne (courbe ou droite) qui sépare les cygnes des oies. À partir des exemples connus, il aura alors induit une loi générale : pour tout oiseau observé qui se place « sous » cette ligne, il sera décidé qu'il s'agit d'un cygne, d'une oie sinon. Mais on peut tracer une infinité de telles lignes. C'est ici que l'apprenant doit préciser le type des connaissances à acquérir, le type du concept à apprendre, en l'espèce quelle est la forme générale de la ligne.

Si l'apprenant impose que la ligne soit droite, le but de l'apprentissage sera de trouver la meilleure ligne droite, en optimisant d'un critère dont il est maître. On remarque d'ailleurs qu'aucune droite ne sépare parfaitement les exemples, mais c'est le prix à payer pour un concept aussi simple. Sur la figure 0.2 est montrée la règle de décision que notre débutant en ornithologie

peut raisonnablement produire. S'il n'impose pas de restriction aussi stricte sur la forme de la ligne, il pourra obtenir une décision comme celle de la figure 0.3.

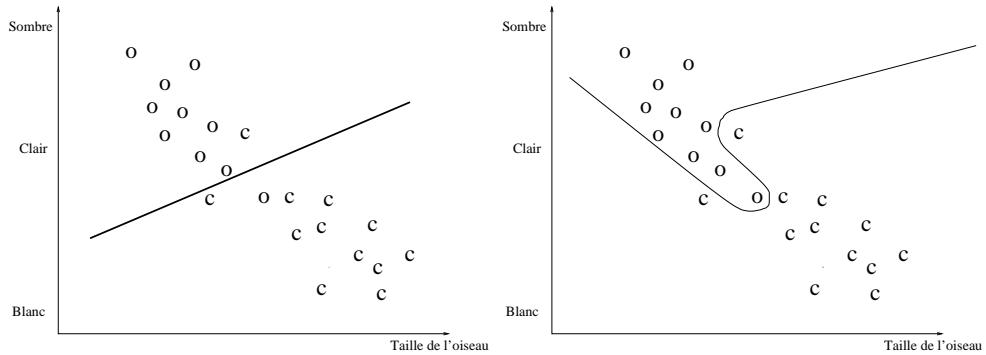


FIG. 0.2 – *Une règle de décision simple et une règle de décision complexe pour séparer les oies des cygnes.*

Quand le brouillard se lève, d'autres oiseaux deviennent visibles. L'apprenant peut alors vérifier la qualité de la règle qu'il a apprise, toujours avec l'aide de son professeur. Dans l'exemple donné sur la figure 0.3, il est facile de constater que la droite qu'il a choisie mène à une erreur environ une fois sur cinq². Pas trop mal, pour un débutant ! Il est assez facile de transposer cet

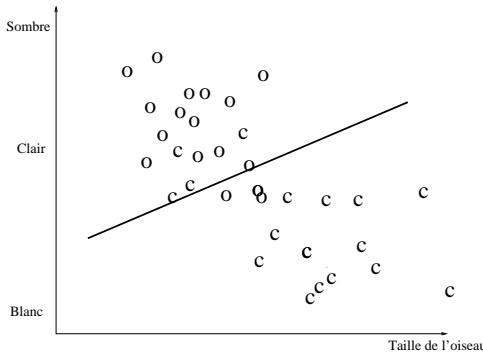


FIG. 0.3 – *Le test de la règle simple sur d'autres oiseaux.*

exemple à l'écriture d'un programme d'apprentissage. Remarquons bien qu'un tel programme n'apprend pas tout court mais apprend quelque chose, en l'occurrence une règle de décision sous la forme d'une équation de droite dans le plan. Cet exemple est caractéristique de ce que font les programmes de reconnaissance des formes. Ce type d'apprentissage par généralisation est d'une immense importance méthodologique et pratique.

Un exemple linguistique

Maintenant, un autre exemple. Supposons que nous disposions d'un ensemble de phrases d'une certaine langue. Est-il possible d'écrire un programme pour en apprendre automatiquement la grammaire ? Pour une langue naturelle, le problème est certainement complexe, ne serait-ce que parce qu'un très grand nombre d'exemples est nécessaire. Mais on peut essayer de le résoudre

2. La ligne courbe donnerait une erreur encore plus grande, nous reviendrons sur ce phénomène au chapitres 2 et 3 .

dans le cas d'un langage artificiel comme ceux qui servent à interroger les bases de données ou pour un sous-ensemble bien délimité de langage naturel. Le langage des échanges entre un client et un employé d'agence de voyage en est un exemple.

Dans de tels cas, il est effectivement possible d'apprendre une grammaire. Il faut cependant imposer au programme des restrictions sur le type de la syntaxe que l'on cherche. L'espace de représentation est ici l'ensemble de toutes les séquences de mots possibles, dont on ne connaît que certaines, linguistiquement correctes. Mais comment définir la grammaire à apprendre? On verra au chapitre 7 que si on oblige cette grammaire à être un automate fini, on peut démontrer que tous les automates finis qui sont compatibles avec les exemples forment un ensemble limité et structuré par une relation d'ordre. Le programme d'apprentissage a alors pour tâche de chercher le meilleur automate dans cet ensemble structuré, encore une fois au sens d'un critère à lui préciser. Remarquons encore que le programme n'apprend pas tout court, mais apprend quelque chose: en l'espèce, une grammaire représentée par un automate fini.

Un exemple d'extraction de connaissances

Une compagnie d'assurances cherche à lancer un nouveau produit, destiné à couvrir le risque de vol d'objets de valeur à domicile. Elle veut faire une campagne de publicité ciblée auprès d'une partie de ses clients. Cette compagnie ne dispose que de peu de produits du même type et par conséquent sa base de données ne comporte qu'une petite proportion d'enregistrements où un client est déjà associé à une assurance contre le vol à domicile. De plus, comme ces clients possèdent déjà un produit analogue, ce n'est pas vers eux qu'il faut principalement cibler la campagne. Mais comment savoir si un client qui n'a pas encore d'assurance de ce type sera intéressé par le nouveau produit?

Une solution est de chercher un profil commun aux clients qui se sont déjà montrés intéressés par un produit de ce type pour viser parmi tous les clients ceux qui ont un profil analogue. Que sera un tel profil? Dans la base de données, chaque client est décrit par un certain nombre de champs, que l'on peut supposer binaires. Par exemples: « âge inférieur à trente ans », « possède une maison », « a un ou plusieurs enfants », « vit dans une zone à risque de vol », etc. Certains champs peuvent être non remplis: les clients qui ont seulement une assurance automobile n'ont pas été interrogés à la constitution de leur dossier sur l'existence d'un système d'alarme dans leur appartement.

Une façon de constituer un profil consiste à découvrir des associations dans les données, c'est-à-dire des implications logiques approximatives. Disons par exemple que la plupart des clients qui possèdent déjà une assurance contre le vol d'objets de valeur à domicile sont plutôt âgés et n'ont en général qu'une voiture, mais haut de gamme. Il semble raisonnable de démarcher parmi tous les clients ceux qui répondent au même profil. L'hypothèse est donc que posséder une seule voiture (mais de luxe) et être d'âge mûr est un profil qui implique sans doute la possession à domicile d'objets de valeur.

Ce type d'apprentissage relève de l'extraction de connaissances et de l'apprentissage non supervisé. Ce dernier terme veut dire que le programme d'apprentissage se débrouille sans professeur: l'expertise est présente dans les données, mais de manière implicite, c'est au programme de la découvrir et de l'utiliser. La combinatoire sous-jacente à ce type d'apprentissage est évidemment très importante.

Organisation et plan de l'ouvrage

L'organisation de cet ouvrage va se décrire maintenant à partir de quelques remarques sur les exemples précédents. Nous disons que, pour le premier, l'espace de recherche est peu structuré. Pourquoi cela ? Parce qu'on ne peut pas dire si telle droite est meilleure que telle autre sans les tester toutes les deux explicitement sur les données. Il s'agit de ce qu'on appelle en général un problème d'optimisation. En revanche, dans le second exemple, nous avons parlé d'une relation d'ordre entre deux solutions, intimement liée à leur qualité relative. Dans ce cas, une exploration partielle de l'ensemble des solutions est possible ; elle sera guidée à la fois par sa structure algébrique et par les données, exemples et contre-exemples, alors que seules les données pouvaient conduire le programme dans le premier cas. Pour le troisième exemple, il n'y a même pas de guidage dans l'espace de recherche par des exemples et des contre-exemples.

Ces remarques sont cruciales pour la conception des algorithmes. C'est la raison pour laquelle nous avons choisi d'organiser le présent ouvrage selon le critère suivant : nous traitons les méthodes d'apprentissage en commençant par celles pour lesquelles l'espace de représentation des concepts à apprendre est fortement structuré, puis de celles pour lesquelles cette hypothèse doit être affaiblie et enfin de celles pour lesquelles l'information *a priori* sur la nature des concepts à apprendre est très faible ou nulle.

Partie 1 : Les fondements de l'apprentissage

Une partie de fondements méthodologiques est d'abord nécessaire. Nous y faisons une présentation générale de la problématique de l'apprentissage et nous donnons les définitions de base (**Chapitre 1 : De l'apprentissage naturel à l'apprentissage artificiel**). Le chapitre suivant propose une introduction aux théories de l'apprentissage par généralisation (**Chapitre 2 : Le problème de l'induction et les grands principes inductifs : une première approche**). Un approfondissement de ce thème sera fait au chapitre 17. Le chapitre suivant traite de la représentation des données et des connaissances et des types d'algorithmes qui sont mis en jeu par la suite (**Chapitre 3 : L'environnement méthodologique de l'apprentissage**).

Partie 2 : Apprentissage par exploration

Nous analysons dans la deuxième partie les méthodes d'apprentissage quand les représentations des concepts forment des ensembles fortement structurés. Nous l'avons appelée *l'apprentissage par exploration*. On y trouve d'abord une méthode très générale (**Chapitre 4 : Induction et relation d'ordre : l'espace des versions**), puis un chapitre sur l'apprentissage dans la logique des prédicats (**Chapitre 5 : La programmation logique inductive**). Le chapitre suivant complète ce point de vue en montrant comment modifier des concepts dans des espaces structurés (**Chapitre 6 : La reformulation et le transfert des connaissances**). Le chapitre suivant (**Chapitre 7 : L'inférence grammaticale**) traite de l'apprentissage des automates et des grammaires. Enfin les méthodes d'apprentissage par évolution simulée, fondées sur l'exploration par algorithmes génétiques, sont exposées (**Chapitre 8 : L'apprentissage par évolution simulée**).

Partie 3 : Apprentissage par optimisation et interpolation

Dans la troisième partie, les connaissances sur la structure des espaces sont plus faibles. Il s'agit de *l'apprentissage par optimisation*. On y trouve le problème de l'apprentissage de droites, mentionné ci-dessus, et leur généralisation à des hyperplans (**Chapitre 9 : L'apprentissage de surfaces séparatrices linéaires**). Une extension désormais classique mène aux réseaux

connexionnistes multicouche (**Chapitre 10 : L'apprentissage de réseaux connexionnistes**). Nous considérons dans la même famille l'apprentissage des arbres de décision et les techniques de combinaison de classificateurs dont l'optimisation est cependant de nature assez différente (**Chapitre 11 : L'apprentissage par combinaison de décisions**). Le fonctionnement et l'apprentissage des réseaux de probabilités conditionnelles est ensuite abordé (**Chapitre 12 : L'apprentissage de réseaux bayésiens**). Le chapitre suivant traite comme le chapitre 7 de l'apprentissage de certaines machines à produire des séquences, mais sous un aspect probabiliste (**Chapitre 13 : L'apprentissage de modèles de Markov cachés**).

Partie 4 : Apprentissage par approximation

La dernière partie, intitulée *l'apprentissage par approximation*, traite des méthodes les moins informées, celles où l'espace des concepts recherchés possède le moins de propriétés. Nous décrivons d'abord les techniques d'apprentissage de règles de classification par des méthodes statistiques qui cherchent à approcher la règle de décision bayésienne. Ce chapitre inclut aussi certains aspects de l'apprentissage par analogie, en particulier la méthodes des plus proches voisins (**Chapitre 14 : L'approximation de la règle de décision bayésienne**).

Dans le chapitre suivant, on s'intéresse aux données non étiquetées par un expert : il s'agit de les organiser et d'y découvrir des régularités et des associations (**Chapitre 15 : La classification non supervisée et la découverte automatique**). Le chapitre suivant s'intéresse aussi à un apprentissage numérique de type « punition » ou « récompense », typiquement applicable à l'apprentissage de son comportement par un robot (**Chapitre 16 : L'apprentissage par renforcement**).

Partie 5 : Approfondissements et annexes techniques

Ce livre se termine par des approfondissements et par la présentation de certains points techniques qui sont énoncés sans démonstration dans les chapitres précédents, souvent à plusieurs occasions. Nous revenons d'abord sur les théories de l'apprentissage par généralisation (**Chapitre 17 : Les grands principes inductifs : approfondissements**). Parmi les annexes, celles qui traitent de l'algorithme **estimation-maximisation** et de l'**optimisation par gradient** sont sans doute les plus référencées dans les chapitres précédents.

Le livre se conclut par une **bibliographie** découpée en deux parties : la première donne des références générales recommandées pour leur qualité et leur accessibilité, essentiellement des livres. La seconde liste donne les références des autres livres, des articles et des rapports cités dans le texte.

Guide de lecture

Après plus de quarante ans de recherches et de réalisations en apprentissage artificiel, il est difficile pour un non initié de savoir comment aborder ce domaine et comment s'y orienter. Nos collaborations avec des utilisateurs des techniques d'apprentissage et avec des chercheurs d'autres disciplines, comme notre activité d'enseignement et d'encadrement avec nos étudiants, nous ont amplement montré l'intérêt d'un ouvrage d'introduction cohérent, articulé autour de grandes lignes directrices.

Il existe déjà des livres d'enseignement et de recherche sur pratiquement chaque sujet abordé dans les chapitres de ce livre et nous ne manquons pas d'y faire référence. Mais si la somme des connaissances qui s'y trouvent est au total bien supérieure à celle contenue dans notre livre, leur lecture conduit à des contradictions dans les notations, à des approfondissements théoriques

de niveau très variable, à des analyses différentes du même problème et à des présentations redondantes voire contradictoires des mêmes sujets.

Il nous a donc paru que la discipline de l'apprentissage artificiel pouvait être présentée de manière unifiée, dans un but d'abord didactique. Ce souci commande le fond comme la forme de cet ouvrage.

Compte tenu de la variété technique des sujets abordés et de l'intérêt personnel de chaque lecteur (autodidacte, enseignant ou étudiant), des parcours différents peuvent être suivis pour la lecture de cet ouvrage.

Nous proposons en particulier les itinéraires suivants, mais ce n'est pas exclusif:

1. Pour une **vue d'ensemble** sur l'apprentissage artificiel, un rapide aperçu des méthodes et un point de vue sur leurs applications:
chapitres 1, 2 (paragraphes 2.1 et 2.2), 3 et 4.
2. Pour un point de vue approfondi sur les **principes méthodologiques**, en particulier statistiques de l'apprentissage:
chapitres 1, 2, 3, 8, 9, 11, 14, 15 et 17.
3. Pour comprendre l'apprentissage de phénomènes essentiellement numériques et pour les applications à la **reconnaissance des formes**:
chapitres 1, 2 (paragraphes 2.1 et 2.2), 3, 9, 10, 11, 13, 14, 15 et 16.
4. Pour acquérir un point de vue sur l'apprentissage dans les **systèmes experts** et le traitement des **données symboliques**:
chapitres 1, 3, 4, 5, 7, 8, éventuellement 6, 12, 14 et éventuellement 16.
5. Pour qui veut réaliser l'apprentissage de concepts à partir de **données séquentielles** (signaux, textes, etc.):
chapitres 1, 3, 5, 6, 11 et 13.
6. Pour qui s'intéresse plus à la **robotique**, l'apprentissage de **comportement**:
chapitres 1, 3, 7, 12, 13, 14 et 16.
7. Pour qui s'intéresse à la **fouille de données**, à l'**extraction de connaissances**:
chapitres 9, 10, 11, 12, 14 et 15.

La situation de l'apprentissage dans l'intelligence artificielle

La figure 0.4 présente la situation des techniques d'apprentissage dans le paysage de l'intelligence artificielle et ses relations avec les concepts clé de cette discipline: représentation des connaissances, raisonnement, type d'algorithme. Cette figure permet en particulier de faire le point sur l'avancement de l'apprentissage dans la perspective de l'ouvrage classique édité par Schavlik et Dietterich ([SD90]).

Les applications industrielles de l'apprentissage artificiel à l'extraction de connaissances des données

Nous avons dit plus haut, sans détailler, que l'extraction de connaissances des données (ECD) a résolu les deux problèmes industriels principaux de l'analyse des données : le fait que le client est souvent imprécis dans la définition du problème qu'il se pose et le fait que les données dont il dispose sont souvent de qualité discutable. Ce paragraphe reprend ce point plus en détail avant de faire un panorama de l'état de l'art de l'industrie ECD.

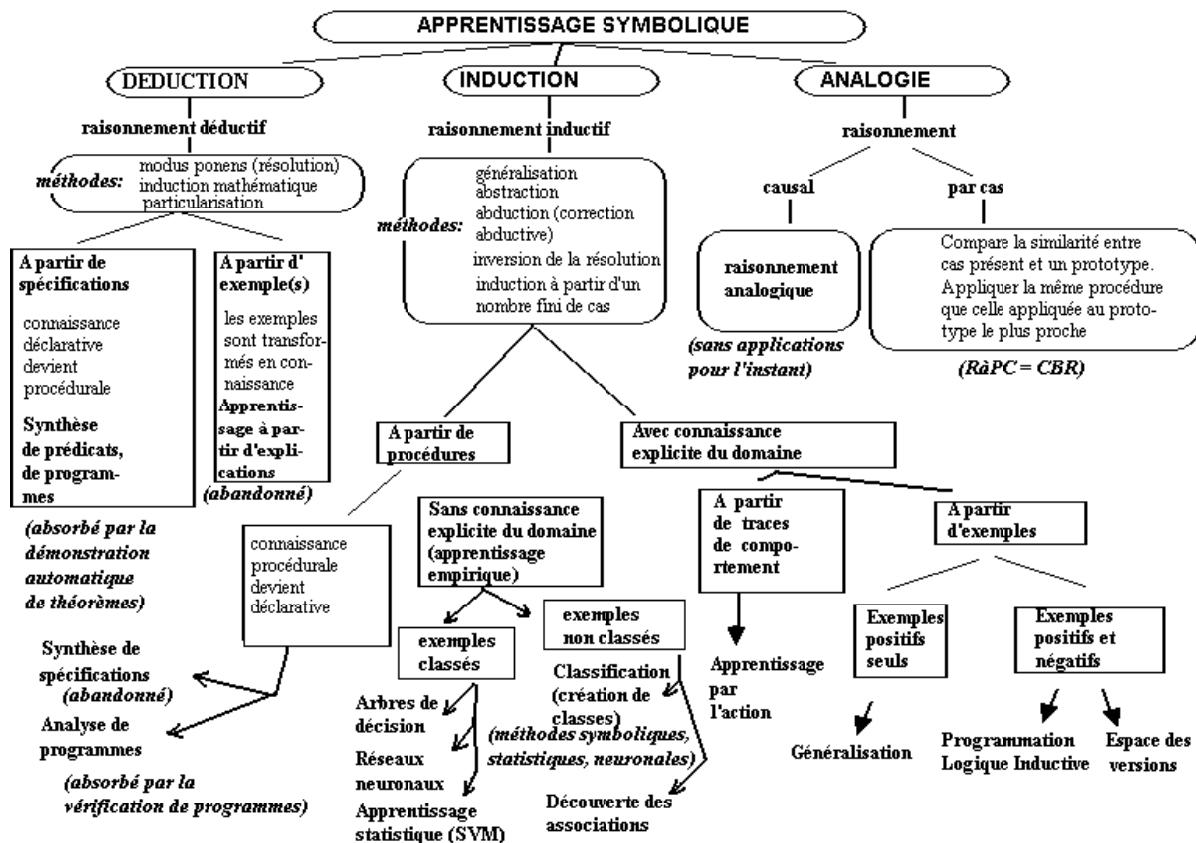


FIG. 0.4 – L'apprentissage artificiel symbolique du point de vue de l'intelligence artificielle.

Intelligibilité des modèles, nettoyage des données

La solution technique apportée par l'ECD au premier problème est liée à l'*intelligibilité* du modèle construit automatiquement. Un système de fouille de données produit des connaissances qui peuvent se trouver sous des formes variées selon la méthode d'apprentissage utilisée : des arbres de décision (chapitre 11), des règles d'association (chapitre 15), des probabilités conditionnelles (chapitres 12 et 14), etc. Nous illustrons ce point dans la suite de ce livre, pour chacune des méthodes décrites. Dans tous les cas, il est capital que le client comprenne parfaitement bien le sens des modèles fournis, car il apprend, en même temps que de nouveaux modèles lui sont décrits, l'information implicite contenue dans ses données, et il est le seul à pouvoir décider de ce qu'est une information intéressante. L'expérience quotidienne de l'analyste de données est qu'il produit souvent des quantités de connaissances encore plus importantes que les données dont il est parti. Posséder une définition précise des façons de sélectionner ce qui intéressant est par conséquent capital. Mais, comme nous venons de la souligner, seul le client est capable de savoir le type des connaissances qui lui manquaient jusqu'alors et que la fouille de données est à même de lui fournir. L'ensemble du procédé d'extraction de connaissances est donc une boucle au sein de laquelle le client joue le rôle de biais, permettant de conserver certaines connaissances et d'en rejeter d'autres. Au cours de cette boucle, le problème que se pose le client s'affine pour converger (on l'espère) vers un problème bien posé.

Le second problème est lié à ce que l'on appelle le *nettoyage* des données, sur lequel nous reviendrons aussi plus en détail au chapitre 3. Divers cas peuvent se combiner : les données sont

sujettes à des bruits de mesure, elles contiennent des points aberrants, les variables de description se recouvrent partiellement ou dépendent l'une de l'autre, etc.

D'après les données sur l'ECD industrielles présentées par G. Piatetsky-Shapiro sur le site Web www.kdnuggets.com environ quinze compagnies se consacrent presque exclusivement au problème du nettoyage des données. Il est facile de constater également que chacune des cent cinquante compagnies environ présentées sur ce site aborde ce problème. Il est par ailleurs frappant de remarquer que la compagnie qui a gagné les deux premières compétitions organisées par le congrès international KDD, en 1995 et 1996, vantait ses méthodes de préparation des données plutôt que la puissance de ses algorithmes d'induction.

Les conseils de la société TMOracle

Les deux problèmes de l'intelligibilité et du nettoyage sont présents dans les conseils que donne la compagnie TMOracle (voir les points 11 et 12 ci-dessous), bien connue pour ces systèmes de gestion de bases de données, et qui a développé ces dernières années, le logiciel *Oracle Data Mining Suite*. C'est maintenant une des offres importantes de cette compagnie. Sur son site, cette compagnie fournit « douze conseils pour le succès en ECD » qui résument son expérience en ce domaine. Voici l'ensemble des conseils de la société Oracle.

1. Extraire à partir d'encore plus de données

Il ne serait pas absurde que pour répondre à l'accroissement des données stockées sous forme électronique, l'ECD dirige son effort en direction du développement de méthodes de plus en plus efficaces d'échantillonnage. Sans que cette option soit négligée, il est frappant de constater que la majorité des outils ne fassent, en quelque sorte, que se résigner à recourir à l'échantillonnage. En fait, les logiciels industriels se vantent de leur rapidité et de leur capacité à traiter des masses énormes de données plutôt que de l'usage d'un procédé d'échantillonnage raffiné. En parallèle, la recherche universitaire montre une attitude assez comparable. Tout se passe comme si on voulait soutenir que toutes les données sont significatives et qu'aucune ne doit être négligée. Cette attitude est d'ailleurs assez raisonnable, dans la mesure où les méthodes d'entrepôts de données permettent déjà d'effectuer des opérations sur les données (par exemple des moyennes) qui reviennent à une forme d'échantillonnage.

2. Créer de nouvelles variables pour mieux faire parler les données

La création de nouvelles variables, de nouveaux attributs, pour mieux décrire la situation est un problème classique(chapitre 3). On l'appelle parfois « induction constructive » en apprentissage automatique. Un autre problème universitaire classique est celui dit de « sélection des attributs » (chapitre 3), où on ne crée pas de nouvelles variables, mais où on tente d'éliminer les attributs inutiles. Les résultats industriels et universitaires convergent : la meilleure façon d'améliorer à la fois précision et intelligibilité est de proposer des combinaisons astucieuses des variables de départ. Par exemple, sur des données bancaires, au lieu de laisser séparées les variables décrivant le revenu et la situation familiale d'un client, il est bien plus intéressant de créer une nouvelle variable combinant son revenu et sa situation familiale de façon à rendre compte directement de ses revenus effectifs.

3. Utiliser une stratégie « en surface d'abord »

Ceci n'est pas un conseil pour rester superficiel, mais pour rechercher d'abord les modèles ou les formes les plus évidents dans les données, avant de commencer à rechercher des modèles profonds et complexes. De toute façon, il est bon de savoir en priorité si les modèles simples engendrés correspondent ou non aux besoins du client.

4. Construire rapidement plusieurs modèles explicatifs.

Même commentaire que pour 3 et voir 12 ci-dessous.

5. Oublier les pratiques traditionnelles d'hygiène en matière de données

Là encore, le conseil est de chercher à accélérer le processus plutôt que d'en assurer la validité avant la création du modèle: les procédures de validation sont mises en route à la fin du processus et non préalablement à celui-ci.

6. Enrichir les données de données extérieures

Il est en effet possible que des descripteurs significatifs aient été oubliés dans une première passe de définition du problème. Il est important de remédier à ces éventuels oublis.

7. Segmenter d'abord les clients et construire des modèles multibuts

La segmentation, appelée classification non supervisée dans le vocabulaire universitaire (chapitre 15), consiste à créer des classes permettant de regrouper les objets étudiés les plus semblables. Il est probable que le problème posé par le client n'ait aucune solution générale (sinon, il l'aurait déjà trouvée!) et que seule une décomposition du problème en sous-problèmes permette de trouver une solution. La segmentation, et chaque essai différent de segmentation, est une façon particulièrement efficace de créer des sous-problèmes. En effet, des formes cachées au sein des données et qui ne sont pas significatives pour l'ensemble des données peuvent le devenir pour une sous-classe.

Un exemple frappant de ce phénomène se rencontre en détection de fraude. L'ensemble des sujets étudiés contient généralement très peu de fraudeurs et les formes caractéristiques de la fraude sont noyées dans celles de la non-fraude. Une segmentation préalable permet de distinguer les non-fraudeurs évidents des soupçonnables de fraude et c'est au sein de cette dernière classe qu'on va chercher à détecter les vrais fraudeurs.

8. Construire automatiquement les modèles

Ce conseil est équivalent à: « utiliser des techniques inductives pour la création de modèles, c'est-à-dire des algorithmes d'apprentissage artificiel ». Dans le contexte de l'ECD dont c'est un des rôles, ce conseil peut sembler évident. Dans le contexte industriel en général, les méthodes de l'apprentissage artificiel sont encore considérées avec une certaine suspicion, en particulier du fait de leur nouveauté. La compagnie Oracle conseille pourtant d'utiliser ces méthodes dont l'étude est le sujet ce livre.

9. Varier les sources de données

Ce conseil est en quelque sorte symétrique du conseil 6. Celui-là venait du soupçon que des descripteurs aient pu être oubliés, celui-ci vient du soupçon que des formes puissent être dissimulées de façon irrécupérable dans certaines données, mais pas dans toutes les données. Quand une forme a été repérée comme importante pour certaines données, il est toujours plus facile de vérifier si elle l'est aussi pour d'autres données.

10. Interpréter les résultats en termes du domaine d'application par une méthode de rétro-ingénierie

Ce conseil est certainement un des plus significatifs et des plus difficiles à implémenter. Les utilisateurs de l'ECD réclament constamment plus d'intelligibilité des connaissances extraites. Inversement, un souci constant des créateurs d'outils est de faire des outils performants et justifiés. Les outils de création de modèles, en particulier les outils statistiques, se sont développés selon une logique qui ne poussait pas à l'intelligibilité de leurs résultats. Les chercheurs qui veulent aujourd'hui faciliter leur usage par une plus grande intelligibilité se heurtent à des problèmes difficiles. Au lieu de recréer des outils produisant des résultats directement intelligibles (ce qui est d'ailleurs une tâche convenant mieux à la re-

cherche universitaire), Oracle conseille de développer des outils de rétro-ingénierie aidant l'utilisateur à récrire le modèle obtenu dans son propre langage.

11. Compléter les données manquantes

Ce problème est à la fois classique et profond. Il relève du nettoyage de données dont nous avons déjà parlé. Chaque système existant comporte une méthode pour prendre en compte les données manquantes. À notre connaissance, il n'existe pas d'outil analysant la nature du manque et proposant une solution adaptée à ce manque. Par exemple, des données peuvent être manquantes parce que le champ est extrêmement difficile ou coûteux à mesurer. Ce cas est différent de celui où les données manquent parce que le champ a été mal défini (comme le champ « cancer de la prostate » dans une base de données comportant des femmes). Le conseil d'Oracle est pragmatique. Il serait étonnant que toutes les causes possibles de données manquantes se trouvent au sein des données d'un client particulier. Il est donc plus sain d'étudier ce problème au cas par cas, et d'appliquer des solutions particulières.

12. Utiliser plusieurs modèles de prédition à la fois

La recherche universitaire a créé de nombreuses procédures qui utilisent plusieurs modèles à la fois, comme les « forêts » de décision (qui comportent plusieurs arbres de décision), le *bagging* et le *boosting* (voir le chapitre 11) qui utilisent des procédures de vote pour choisir le modèle qui sera appliqué sur une donnée nouvelle. Le but de ces procédures est en général d'améliorer la précision des prévisions du modèle. Mais ce n'est pas ce que signifie le conseil d'Oracle. Il s'agit ici plutôt d'améliorer la compréhension du client en lui présentant plusieurs modèles, chacun d'entre eux soulignant un aspect des données, au lieu de présenter un seul modèle mélangeant, de façon peu compréhensible, plusieurs points de vue différents.

Nous allons maintenant étudier les outils, les méthodes et les besoins du marché de l'ECD. Pour ceci, nous allons analyser les résultats d'enquêtes menées par G. Piatetsky-Shapiro (GPS) sur son site dédié au marché industriel de l'ECD.

Les outils utilisés par les compagnies couvrant une part appréciable du marché

Au-delà de la diversité des approches des diverses compagnies, on peut noter qu'elles ont deux traits en commun.

- Le premier est que toutes les compagnies se vantent de l'intelligibilité de leurs résultats, quelles que soient les méthodes inductives (c'est-à dire les algorithmes d'apprentissage) utilisées. Cette constance souligne l'importance capitale de l'intelligibilité des résultats obtenus sur le marché de l'ECD.
- Le second point présente deux versants complémentaires :
 - d'une part, chaque compagnie possède une méthode d'induction phare qu'elle prétend être supérieure aux autres ;
 - d'autre part, il existe peu de compagnies qui ne proposent pas un éventail de produits variés.

L'exemple de la société CART est typique. Cette compagnie, dédiée au départ aux arbres de régression (le RT de CART signifie *regression trees*), offre maintenant des arbres ou des règles de décision (chapitre 11), des réseaux connexionnistes (chapitre 10), des réseaux bayésiens (chapitre 12) et des réseaux de dépendance, des machines à vecteurs supports (SVM, chapitre 9), des approches utilisant la théorie des ensembles approximatifs (*rough sets*) et les approches par algorithmes génétiques (chapitre 8).

Compagnie	Outils de base	Oct 99	Ju 00	Oct 01	Plan 02	% St/Sy
SPSS Clementine	AD, RN St, ass	17 %	20 %	18 %	13 %	St : 9 % Sy : 9 %
Megaputer	symb	1 %	1 %	6 %	13 %	Sy : 6 %
SAS	st	26 %	32 %	18 %	18 %	St : 18 %
Urban Science Gain-Smarts	rég	5 %		1 %	12 %	St : 1 %
MINEit Easy-Miner	?				6 %	non sig
Angoss	RN, rég χ^2 , AD	3 %	7 %	3 %	5 %	St : 2 % Sy : 1 %
Oracle Darwin	ass + sg	2 %		2 %	5 %	Sy : 2 %
IBM Iminer	ass, sg sq, cl	7 %	9 %	3 %	4 %	St : 1 % Sy : 2 %
Business Objects Alice	A.Dec				3 %	
SGI Mineset	A.D, V	4 %	5 %	2 %	3 %	Sy : 2 %
SPSS Answer-Tree	AR	15 %		16 %	2 %	St : 16 %
CART/ MARS	AR	6 %	7 %	11 %	3 %	St : 11 %
Model 1				1 %	2 %	
Quadstone		1 %			1 %	
Statistica	St	2 %		2 %	1 %	St : 2 %
Wizsoft		1 %			1 %	
WhiteCross					0 %	
Xaffinity	1 %			0 %	0 %	
Autres		8 %	20 %	11 %	7 %	12 % non sig.
Total: non significatif du point de vue % St/Sy						18 %
Total : outil plutôt statistique						60 %
Total : outil plutôt symbolique						22 %

TAB. 0.1 – Les outils utilisés par les entreprises en ECD

Légende AD = arbres de décision ; V = méthodes de visualisation ; ass = règles d'association ;

AR = arbres de régression ; rég = méthodes de régression ; st = outils statistiques et d'analyse des données, y compris régression ; RN = réseaux connexionnistes ;

symb = méthodes de nature symbolique non expliquées en détail par l'industriel ; χ^2 = tests du χ^2 ;

sq = utilisation de séquences ;

sg = segmentation, c'est-à-dire classification automatique ;

cl = méthodes de classification ; ? = totalement obscur ;

non sig = non significatif pour cet indice.

On constate dans le tableau 0.2 que les outils statistiques sont utilisés à environ 60 % et les outils symboliques à environ 40 %. Dans le tableau 0.2, la proportion est un peu plus faible pour les outils symboliques : ils sont un peu moins vendus qu'utilisés. Ceci étant, il reste que l'ECD est une grande consommatrice d'outils statistiques et les techniques de retro-ingénierie semblent compenser leur relative difficulté d'usage.

Les enquêtes de GPS sont relatives à des compagnies plutôt qu'à des outils proprement dits. La colonne « outils de base » du tableau 0.1 est donc issue de nos propres connaissances sur les outils vendus par ces compagnies, quand cette information peut être obtenue.

Afin d'évaluer les parts de marché respectives des outils statistiques et des outils symboliques nous avons aussi évalué en gros cette part pour chacune des compagnies. Le symbole *St* désigne un outil plutôt statistique et le symbole *Sy* un outil plutôt symbolique. Le symbole $\% St/Sy$ désigne une évaluation du rapport entre outils statistiques et outils symboliques. L'ensemble de l'enquête est résumé dans le tableau 0.1.

Réseaux connexionnistes	13 %	St: 13 %
Arbres de décision /Règles	19 %	Sy: 19 %
Régression Logistique	14 %	St: 14 %
Statistiques	17 %	St: 17 %
Bayésien	6 %	St: 3 % Sy: 3 %
Visualisation	8 %	St: 4 % Sy: 4 %
Règles d'association	7 %	Sy: 7 %
Méthodes hybrides	4 %	St: 2 % Sy: 2 %
Fouille de textes (non significatif)	2 %	ns: 2 %
Fouille de la toile (non significatif)	5 %	ns: 5 %
Agents (non significatif)	1 %	ns: 1 %
Autres (non significatif)	4 %	ns: 4 %
Total: non significatif	12 %	
Total: outil plutôt statistique	53 %	
Total: outil plutôt symbolique	35 %	

TAB. 0.2 – *Enquête de GPS: « Quelles techniques d'ECD utilisez-vous régulièrement? » (Août 2001)*

Les domaines d'application des méthodes inductives de fouille de données

Le tableau 0.3 combine une enquête de GPS (« À quoi avez-vous appliqué l'extraction de connaissances? ») et notre estimation en 1998. Cette dernière calcule le pourcentage de compagnies déclarant travailler sur une application. Une compagnie peut être comptée plusieurs fois pour des applications différentes.

On remarque l'accroissement des applications bancaires, du commerce électronique et des télécommunications, ainsi que la décroissance des applications commerciales classiques. Les quelques rubriques représentées en 1998 qui ont disparu sont dans le tableau 0.4.

L'attrition³ était plus connue en France sous le nom de *churn* (barattage); il semble que les

3. Il s'agit du problème des abonnés fuyants. Voir aussi au chapitre 3, page 117.

Application	1998 (estimation)	mai 2001	plans pour 2002
Banque	6 %	17 %	13 % (++)
Biologie/Génétique	8 %	8 %	8 %
Vente directe/Relations clientèle	22 %		11 % (- -)
Commerce électronique/Toile	6 % / 0 %	15 %	10 % (++)
Jeux	1.5 %		1 %
Détection des fraudes	10 %	8 %	11 %
Assurance	4 %	6 %	6 %
Investissement/Gestion des stocks	9 %	4 %	4 % (- -)
Fabrication			4 %
Produits pharmaceutiques	5 %	5 %	6 %
Vente au détail	15 %	6 %	6 % (- -)
Science		8 %	6 %
Sécurité			2 %
Télécommunications	2.5 %	11 %	8 % (++)
Autres	11 %	11 %	5 %

TAB. 0.3 – *Les domaines d'application.*

Education	2 %
Lutte contre la criminalité	1 %
Fidélisation / Attrition	8 %

TAB. 0.4 – *Les domaines d'application disparus depuis 1998.*

applications en ce domaine soient terminées.

Environ 50 % des compagnies qui vendent de l'ECD le font grâce à des produits dédiés aux relations clientèle. Ce marché qui a été florissant en 1998 semble se rétrécir rapidement. L'explication de ce phénomène se trouve peut-être dans les résultats de l'enquête suivante de GPS, dont les résultats sont présentés dans le tableau 0.5 : « Quelle est votre expérience d'échecs de projets en relations clientèle » ?

% déclarés comme échec	% d'utilisateurs dans cette tranche	Total partiel
0 - 20 %	11 %	
21 - 40 %	16 %	27 %
41 - 60 %	33 %	33 %
61 - 80 %	29 %	
40 %	81 - 100 %	11 %

TAB. 0.5 – *Le succès en relations clientèle.*

En conclusion, les usagers signalent à peu près 50% de réussites de l'ECD en relations clientèle, ce qui est insuffisant pour assurer le succès commercial de cette approche.

Un dernier facteur relatif aux applications est celui du type des données analysées. C'est ce que détermine cette autre enquête de GPS (tableau 0.6) « Quel type de données avez-vous analysées en 2001 » ?

On remarque que les usagers montrent des besoins importants en fouille de textes, ce qui

Flot des clics sur la Toile	13 %
Contenu de la Toile	14 %
Textes	18 %
Bases de connaissance	8 %
Données XML	8 %
Images	4 %
Données audio/vidéo	2 %
Données de CAO	1 %
Séries temporelles	16 %
Autres données complexes	17 %

TAB. 0.6 – *Les données analysées. Le « flot des clics sur la Toile » désigne le Web clickstream.*

est d'autant plus vrai qu'on y ajoute les analyses du contenu de la Toile qui est une forme de fouille de textes. Ce domaine d'importance industrielle extrême est relativement peu étudié par les spécialistes d'ECD et la plupart des outils utilisés viennent de la communauté du traitement automatique de la langue naturelle.

Le problème de l'intelligibilité

Comme nous l'avons déjà plusieurs fois remarqué, le problème de l'intelligibilité est capital. Les méthodes d'induction ne sont en général pas « naturellement » intelligibles, c'est pourquoi un effort de rétro-ingénierie est toujours nécessaire. Pour illustrer une façon de réaliser cet effort, prenons l'exemple du CoIL Challenge 2000. Il s'agit d'une compétition où on demandait de prévoir et de décrire le comportement d'acheteurs de caravanes à partir d'une base de données. La tâche de prévision était jugée selon des critères de précision et celle de description selon des critères d'intelligibilité.

Le vainqueur de la tâche de prédiction a utilisé un apprentissage bayésien naïf (chapitre 14), ce qui confirme le fait que cette méthode est à la fois simple d'implémentation et d'une grande précision en prédiction. En revanche, elle ne fournit aucune explication quand à la façon dont elle prend des décisions. Les vainqueurs de la tâche de description ont utilisé une intéressante combinaison de méthodes statistiques et symboliques. Les auteurs utilisent des algorithmes génétiques (chapitre 8) pour engendrer la structure d'un réseau connexionniste (chapitre 10). Ils sont moins précis en résultats purs que le vainqueur, mais ils obtiennent au passage de précieuses informations sur les attributs les plus significatifs pour la tâche de classification. Pour l'intelligibilité, c'est-à-dire la tâche de description des consommateurs, ils combinent trois approches. D'abord, ils essaient de créer un grand nombre d'attributs compréhensibles, comme par exemple la taille de la famille, le caractère estimé plus ou moins carriériste des individus, le nombre de bicyclettes dans la famille, etc. Ensuite, ils utilisent un test statistique du χ^2 pour mesurer l'importance relative des descripteurs dans leur contribution à la solution. Enfin, ils engendrent des règles d'association sur les descripteurs significativement reliés à la solution pour conclure sur l'achat d'une caravane. Avec ces restrictions, il s'avère qu'ils obtiennent seulement sept règles ayant un support supérieur à 10 %, ce qui fournit un ensemble bien compréhensible de règles décrivant l'acheteur de caravanes.

Il est remarquable que les statistiques et les réseaux connexionnistes, deux méthodes fondamentalement numériques, soient utilisées ici pour améliorer la compréhension des résultats d'apprentissage de règles symboliques.

Quelques termes et faux-amis anglais-français.

Apprentissage	<i>Learning</i>
Apprentissage artificiel <i>ou</i> automatique	<i>Machine Learning</i>
Apprentissage par cœur	<i>Rote learning</i>
Attribut	<i>Feature ou attribute</i>
Attribut numérique	<i>Continuous valued attribute</i>
Attribut arborescent	<i>Tree-structured attribute</i>
Classification automatique	<i>Clustering</i>
Distributions bienveillantes	<i>Benign distributions</i>
Élagage	<i>Pruning</i>
Escalade	<i>Hill-climbing</i>
Exemple	<i>Example ou instance</i>
Fouille de données	<i>Data mining</i>
Hypothèse cohérente (complète et correcte)	<i>Consistant (complete and correct) hypothesis</i>
Objet <i>ou</i> forme <i>ou</i> observation	<i>Object ou instance</i>
Perte	<i>Loss</i>
Pertinence du principe <i>ERM</i>	<i>Consistency of ERM principle</i>
Plus proche voisin	<i>Nearest neighbor ou neighbour</i>
Produit scalaire	<i>Inner product</i>
Professeur	<i>Teacher</i>
Règle de classification <i>ou</i> de classement	<i>Classification rule ou Classifier</i>
Reconnaissance des formes	<i>Pattern Recognition</i>
Réseau connexionniste	<i>Neural network</i>
Surapprentissage	<i>Overfitting</i>
Performance	<i>Fitness</i>

Notations

Notations générales

P	Une probabilité
p	Une densité de probabilités
\mathbb{N}	L'ensemble des entiers naturels
\mathbb{R}^d	L'espace euclidien de dimension d
$\mathbb{B}^d = \{0, 1\}^d$	L'espace booléen de dimension d
\mathcal{O}	L'ordre de grandeur maximal de complexité d'un algorithme
$\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix}$	Un vecteur
$\mathbf{x}^T = (x_1, \dots, x_d)$	Un vecteur transposé
$\ \mathbf{x}\ $	La norme du vecteur \mathbf{x}
M^{-1}	La matrice inverse d'une matrice carrée M
M^T	La matrice transposée d'une matrice M
M^+	La matrice pseudo-inverse d'une matrice M . Par définition, $M^+ = M^T(MM^T)^{-1}$
$\Delta(\mathbf{x}, \mathbf{y})$	La distance euclidienne entre deux vecteurs \mathbf{x} et \mathbf{y} de \mathbb{R}^d
$\frac{\partial}{\partial x} f(x, y)$	La dérivée partielle par rapport à x de la fonction f des deux variables x et y
$\nabla_{\mathbf{A}} J(\mathbf{A}, \mathbf{B})$	Le vecteur dérivé par rapport au vecteur \mathbf{A} de la fonctionnelle J des deux vecteurs \mathbf{A} et \mathbf{B}

Les éléments en jeu dans l'apprentissage

\mathcal{X}	L'espace de représentation des objets (des formes)
\mathcal{U}	L'espace de supervision (des sorties désirées)
\mathcal{S}	L'échantillon d'apprentissage (un ensemble ou une suite d'exemples)
\mathcal{S}_+	Les exemples positifs
\mathcal{S}_-	Les exemples négatifs
\mathcal{A}	L'échantillon d'apprentissage quand on divise \mathcal{S} en \mathcal{A} , \mathcal{T} et \mathcal{V}

\mathcal{T}	L'échantillon de test
\mathcal{V}	L'échantillon de validation
m	La taille d'un échantillon d'apprentissage (le nombre d'exemples)
$\mathbf{z}_i = (\mathbf{x}_i, \mathbf{u}_i)$	Un exemple (élément d'un échantillon d'apprentissage)
\mathbf{x}_i	La description d'un objet dans un espace de représentation
x_{ij}	La valeur de la coordonnée j de la description de l'objet \mathbf{x}_i dans \mathbb{R}^d

Les principes de l'apprentissage inductif

\mathbf{u}_i	La supervision, ou sortie désirée, d'un exemple
$f : \mathcal{X} \rightarrow \mathcal{U}$	La fonction cible (celle que l'on cherche à apprendre)
\mathcal{H}	L'espace des hypothèses d'apprentissage
$h \in \mathcal{H}$	Une hypothèse produite par un apprenant (un algorithme d'apprentissage)
$\mathbf{y} = h(\mathbf{x})$	La prédiction faite par l'hypothèse h sur la description \mathbf{x} d'un exemple
$l(f, h)$	La perte (ou distance) entre la fonction cible et une hypothèse
$R_{Réel}$	Le risque réel
R_{Emp}	Le risque empirique
$h_{\mathcal{H}}^*$	L'hypothèse de \mathcal{H} qui minimise le risque réel
$h_{\mathcal{S}, \mathcal{H}}^*$	L'hypothèse de \mathcal{H} qui minimise le risque empirique sur \mathcal{S}
$h_{algo, \mathcal{S}, \mathcal{H}}^*$	L'hypothèse trouvée par un algorithme ayant \mathcal{S} en entrée et cherchant $h_{\mathcal{S}, \mathcal{H}}^*$ dans \mathcal{H}

L'apprentissage d'une règle de classification

\mathcal{C}	L'ensemble des classes
C	Le nombre de classes
ω_i	Une classe de \mathcal{C}

La logique

$a \wedge b$	a ET b , quand a et b sont des valeurs binaires
$a \vee b$	a OU b
$\neg a$	NON a
$a \rightarrow b$	a implique b

Table des matières

Avant-propos	v
Quelques applications de l'apprentissage artificiel	vi
Quelques définitions de base	vi
Deux champs industriels de l'apprentissage artificiels : la reconnaissance des formes et la fouille de données	vii
Les caractéristiques de l'apprentissage artificiel	viii
Trois exemples d'apprentissage	x
Organisation et plan de l'ouvrage	xiii
Guide de lecture	xiv
La situation de l'apprentissage dans l'intelligence artificielle	xv
Les applications industrielles de l'apprentissage artificiel à l'extraction de connaissances des données	xv
Notations	xxvii
I Les fondements de l'apprentissage	1
1 De l'apprentissage naturel à l'apprentissage artificiel	3
1.1 L'apprentissage artificiel	3
1.2 Deux exemples : apprendre à jouer, apprendre à lire	5
1.2.1 Apprendre à jouer	5
1.2.2 Apprendre à reconnaître des caractères manuscrits	7
1.3 Deux approches : la cybernétique et les sciences cognitives	9
1.3.1 La cybernétique	10
1.3.2 Le pari du cognitivisme	11
1.4 Les concepts de base de l'apprentissage	13
1.4.1 Un scénario de base pour l'induction	13
1.4.2 Quelques notions clés	14
1.4.3 L'induction vue comme une estimation de fonction	18
1.5 L'induction comme un jeu entre espaces	20
1.5.1 L'apprentissage est impossible...	22
1.5.2 ... sans limiter l'espace des hypothèses	23
1.5.3 L'exploration de l'espace des hypothèses	26
1.6 Retour sur l'organisation de l'ouvrage	27

2 Première approche théorique de l'induction	35
2.1 Deux exemples d'induction	37
2.1.1 Le système ARCH	37
2.1.2 Le perceptron	39
2.2 Approche de l'induction	42
2.2.1 Le compromis biais-variance	42
2.2.2 Comment définir formellement le problème de l'induction?	45
2.2.3 Quel principe inductif adopter? Une introduction	46
2.2.4 Comment analyser l'apprentissage?	48
2.3 Analyse dans le pire cas : l'apprentissage <i>PAC</i>	49
2.3.1 Étude des conditions de validité de l' <i>ERM</i>	50
2.3.2 Le cas de la discrimination: l'analyse <i>PAC</i>	53
2.4 Analyse dans un cas moyen : l'analyse bayésienne	57
2.4.1 Nature de l'analyse bayésienne	58
2.4.2 Le risque bayésien et la décision optimale	59
2.4.3 Cas particuliers de la décision bayésienne	60
2.4.4 Panorama des méthodes inductives dans le cadre bayésien	63
2.4.5 Et si l'espace des hypothèses ne contient pas la fonction cible?	63
2.4.6 En résumé : la procédure inductive bayésienne	64
2.5 Discussion : Quels types d'analyses et de principes inductifs?	64
2.6 Les grands principes inductifs avec régulation des hypothèses	65
2.6.1 L'idée générale: le réglage de la classe d'hypothèses	66
2.6.2 La sélection de modèles	67
2.7 Discussion et perspectives	68
2.8 Notes historiques et bibliographiques	69
3 L'environnement méthodologique de l'apprentissage	73
3.1 L'espace des données d'apprentissage	76
3.1.1 La représentation des objets de l'apprentissage	76
3.1.2 Le prétraitement des données	80
3.2 L'espace des hypothèses d'apprentissage	86
3.2.1 Le problème général de la représentation des connaissances	86
3.2.2 La classification	87
3.2.3 La régression	90
3.2.4 Les distributions de probabilités	90
3.2.5 Les arbres de décision	90
3.2.6 Les hiérarchies de concepts	91
3.2.7 Les réseaux bayésiens et les modèles graphiques	92
3.2.8 Les chaînes de Markov et les modèles de Markov cachés	93
3.2.9 Les grammaires	93
3.2.10 Les formalismes logiques	94
3.3 La recherche dans l'espace des hypothèses	96
3.3.1 Caractérisation de l'espace de recherche	96
3.3.2 Caractérisation des fonctions de coût	96
3.3.3 Les méthodes d'optimisation	97
3.4 L'évaluation de l'apprentissage	102
3.4.1 L'évaluation <i>a priori</i> : critères théoriques	103
3.4.2 L'évaluation empirique <i>a posteriori</i> : généralités	104

3.4.3	Risque empirique et risque réel	105
3.4.4	La sélection de modèle en pratique	106
3.4.5	L'estimation du risque réel d'une hypothèse	111
3.4.6	Le réglage des algorithmes par un ensemble de validation	115
3.4.7	D'autres critères d'appréciation	117
3.5	La comparaison des méthodes d'apprentissage	118
3.5.1	La comparaison de deux hypothèses produites par un même algorithme sur deux échantillons de test différents.	119
3.5.2	La comparaison de deux algorithmes sur des ensembles de test différents .	119
3.5.3	La comparaison de deux algorithmes sur le même ensemble de test	120
II	Apprentissage par exploration	123
4	Induction et relation d'ordre : l'espace des versions	125
4.1	Les concepts de base	128
4.1.1	La description des attributs, la description des concepts	128
4.1.2	Les sélecteurs	128
4.1.3	La relation de généralité entre les hypothèses	129
4.1.4	La relation entre un objet et un concept	131
4.2	La structuration de l'espace des hypothèses	132
4.2.1	Préliminaires	132
4.2.2	Un exemple : les paires de rectangles	133
4.2.3	Un ordre partiel sur l'espace des hypothèses	134
4.2.4	Quelques opérateurs de spécialisation et de généralisation	136
4.2.5	Quelques propriétés utiles d'un espace structuré par une relation d'ordre partiel	137
4.3	La construction de l'espace des versions	139
4.3.1	Illustration: retour sur l'exemple des rectangles	139
4.3.2	L'algorithme d'élimination des candidats	140
4.3.3	Deux exemples	140
4.3.4	Un exemple d'application : le système LEX	148
4.4	Analyse de l'algorithme d'élimination de candidats	149
4.4.1	Complexité au pire	149
4.4.2	Le point de vue de l'apprentissage <i>PAC</i>	150
4.5	La représentation des connaissances par un treillis de Galois	151
4.5.1	La construction de la structure	151
4.5.2	L'utilisation pour l'apprentissage	153
5	La programmation logique inductive	157
5.1	La programmation logique inductive: le cadre général	160
5.1.1	Complexité de l'induction et expressivité du langage d'hypothèses	160
5.1.2	La relation de couverture en logique du premier ordre	161
5.1.3	La subsomption en logique du premier ordre	163
5.1.4	Un résumé des relations de subsomption possibles	165
5.2	La logique des prédictats et les programmes logiques: terminologie	166
5.2.1	La syntaxe de la logique des prédictats	166

5.2.2	Système de preuve pour les langages de clauses	168
5.3	La structuration de l'espace des hypothèses en logique des prédicts	170
5.3.1	Le calcul de la <i>lgg</i> pour la θ -subsumption	170
5.3.2	Le calcul de <i>rlgg</i> pour la θ -subsumption relative	172
5.3.3	Le calcul de <i>lgg</i> pour la résolution inverse	174
5.4	L'exploration de l'espace des hypothèses	177
5.4.1	Le squelette des algorithmes de PLI	178
5.4.2	Les biais de recherche dans l'espace d'hypothèses	180
5.5	Deux exemples de systèmes de PLI	182
5.5.1	Un système empirique descendant : FOIL	182
5.5.2	Un système empirique ascendant : PROGOL	184
5.6	Les domaines d'application de la PLI	186
5.7	Les chantiers de la PLI	188
5.7.1	Les problèmes à résoudre	188
6	Reformulation et transfert de connaissances	193
6.1	L'apprentissage en présence de théorie	194
6.2	L'apprentissage par examen de preuve (<i>EBL</i>)	194
6.2.1	Le principe de l' <i>EBL</i>	194
6.2.2	Une illustration de l'apprentissage <i>EBL</i>	195
6.2.3	Discussion sur l'apprentissage de concept à partir d'explications	198
6.2.4	L'apprentissage de connaissances de contrôle à partir d'explications	199
6.2.5	Bilan sur l'apprentissage à partir d'explications	201
6.3	Abstraction et reformulation des connaissances	201
6.4	Changement de repère et raisonnement par analogie	203
6.5	Bilan	205
7	L'inférence grammaticale	207
7.1	Définitions et notations	212
7.1.1	Langages, grammaires, automates et partitions	212
7.1.2	Échantillons d'un langage et automates associés	218
7.2	Les protocoles de l'inférence : quelques résultats théoriques	220
7.2.1	La spécification d'un problème d'inférence grammaticale	220
7.2.2	L'identification à la limite d'une grammaire	221
7.2.3	Deux propriétés de l'identification à la limite.	222
7.2.4	Autres protocoles pour l'inférence de grammaires.	222
7.2.5	L'inférence grammaticale et l'apprentissage <i>PAC</i>	223
7.2.6	Résultats <i>PAC</i> pour les langages réguliers	224
7.2.7	Apprentissage <i>PACS</i> : <i>PAC Simple</i>	225
7.2.8	Apprentissage <i>PAC</i> avec distributions bienveillantes	226
7.3	L'espace de recherche de l'inférence régulière	226
7.3.1	Le point de la situation	226
7.3.2	Deux propriétés fondamentales	227
7.3.3	La taille de l'espace de recherche	228
7.4	L'inférence régulière sans échantillon négatif	228
7.4.1	Une méthode caractérisable : l'inférence de langages k-réversibles	228
7.4.2	Une méthode heuristique: l'algorithme ECGI	230
7.5	L'inférence régulière sous contrôle d'un échantillon négatif	233

7.5.1	L'ensemble frontière	233
7.5.2	Le lien avec l'espace des versions	233
7.5.3	Les algorithmes RIG et BRIG	233
7.5.4	L'algorithme RPNI	234
7.5.5	Variantes et extensions	236
7.6	L'inférence de grammaires algébriques.	237
7.6.1	Présentation	237
7.6.2	L'apprentissage à partir d'échantillons structurés.	237
7.6.3	Les méthodes par exploration.	239
7.6.4	Une méthode avec oracle.	239
7.6.5	L'inférence de grammaires linéaires équilibrées	240
7.7	Quelques extensions	240
7.7.1	Les grammaires stochastiques	240
7.7.2	Le point de vue connexionniste	242
8	Apprentissage par évolution simulée	245
8.1	Trois espaces au lieu de deux	247
8.2	Un modèle formel simplifié de l'évolution	249
8.2.1	Le jeu entre \mathcal{H} et \mathcal{G}	249
8.2.2	L'apprentissage comme processus d'évolution d'une population	249
8.3	Les algorithmes génétiques	250
8.3.1	La représentation dans l'espace génotypique	251
8.3.2	L'algorithme générique	251
8.3.3	L'initialisation de la population	252
8.3.4	Les opérateurs	252
8.3.5	La sélection	255
8.3.6	Le théorème des schémas : une explication de la puissance des AG?	258
8.4	Les stratégies d'évolution	262
8.5	La programmation génétique	262
8.5.1	La représentation des programmes	265
8.5.2	Les opérateurs génétiques sur les programmes	267
8.5.3	Évaluation et sélection	267
8.5.4	Le fonctionnement	267
8.5.5	Illustrations	268
8.5.6	Une analyse de la programmation génétique	271
8.6	La coévolution	272
8.6.1	Un exemple d'écologie : les systèmes de classeurs	274
III	Apprentissage par optimisation	277
9	L'apprentissage de surfaces séparatrices linéaires	279
9.1	Généralités.	281
9.1.1	Hyperplans séparateurs et discriminants dans un problème à deux classes.	281
9.1.2	Un peu de géométrie dans \mathbb{R}^d	282
9.2	L'apprentissage d'un hyperplan pour discriminer deux classes	283
9.2.1	Une solution globale	283
9.2.2	Une méthode itérative: l'algorithme de Ho et Kashyap.	286

9.2.3	Un autre calcul: l'algorithme du perceptron	287
9.2.4	L'hyperplan discriminant de Fisher	290
9.2.5	Surfaces séparatrices non linéaires	292
9.2.6	Et pour plus de deux classes?	292
9.3	Les séparateurs à vastes marges (SVM)	293
9.3.1	La recherche des séparateurs linéaires à vastes marges	294
9.3.2	Quelle justification pour les SVM?	302
9.3.3	La régression par fonctions noyau et exemples critiques	307
9.3.4	Conclusions sur les SVM	309
10	L'apprentissage de réseaux connexionnistes	311
10.1	Les différents éléments d'un réseau connexionniste	313
10.2	L'architecture multicouche	315
10.2.1	La transmission de l'information dans un réseau multicouche	315
10.2.2	Un exemple	316
10.2.3	Un autre exemple: le problème « XOR »	318
10.2.4	Le protocole d'apprentissage	319
10.2.5	Le codage des exemples d'apprentissage	320
10.3	L'algorithme d'apprentissage	320
10.3.1	Retour sur le perceptron	321
10.3.2	L'apprentissage par rétropropagation du gradient de l'erreur	325
10.3.3	L'organisation des calculs	326
10.3.4	Retour sur l'exemple	326
10.3.5	Une variante	328
10.3.6	Quand arrêter l'apprentissage?	329
10.3.7	Le problème des minima locaux	329
10.4	Quelques résultats théoriques sur les réseaux connexionnistes	329
10.4.1	Pouvoir d'expression	329
10.4.2	Complexité	330
10.4.3	Réseaux connexionnistes et apprentissage <i>PAC</i>	330
10.5	Comment choisir l'architecture d'un réseau?	331
11	Apprentissage par combinaison de décisions	333
11.1	Les arbres de décision	335
11.1.1	Principe	335
11.1.2	La construction récursive d'un arbre de décision	336
11.1.3	Comment élaguer un arbre trop précis	343
11.1.4	Un exemple: les iris de Fisher	347
11.1.5	Traduction des arbres de décision en logique des propositions	349
11.2	Les arbres de régression	352
11.2.1	Le principe	352
11.2.2	La construction	353
11.2.3	Un exemple	353
11.2.4	La fin de la construction et l'élagage	354
11.3	Le <i>boosting</i> d'un algorithme d'apprentissage	354
11.3.1	Plusieurs experts valent mieux qu'un	354
11.3.2	Le premier algorithme de boosting	355
11.3.3	Le boosting probabiliste et l'algorithme ADABOOST	357

11.3.4	Les propriétés de l'algorithme ADABOOST	358
11.3.5	L'utilisation du boosting	360
11.3.6	Boosting et théorie <i>PAC</i>	360
11.3.7	Le « bagging »	361
12 L'apprentissage de réseaux bayésiens		363
12.1	Les réseaux d'inférence bayésiens	364
12.1.1	Définitions et notations	366
12.1.2	La d-séparation	366
12.1.3	Définition formelle d'un réseau bayésien	368
12.2	Les inférences dans les réseaux bayésiens	368
12.2.1	Schémas d'inférence	369
12.2.2	La d-séparation généralisée	372
12.3	L'apprentissage des réseaux bayésiens	374
12.3.1	Apprentissage avec structure connue et données complètes	375
12.3.2	Apprentissage avec structure inconnue et données complètes	376
12.3.3	Apprentissage en présence de données incomplètes	379
12.3.4	Apprentissage avec structure connue et données incomplètes	379
12.3.5	Apprentissage avec structure inconnue et données incomplètes	381
13 L'apprentissage de modèles de Markov cachés		385
13.1	Les modèles de Markov observables	388
13.2	Les modèles de Markov cachés (HMM)	389
13.2.1	Définition	389
13.2.2	Pourquoi faut-il des variables cachées ?	389
13.2.3	Notations	391
13.2.4	Deux types de HMM	392
13.2.5	Comment un HMM engendre une séquence	393
13.3	Les HMM comme règles de classification de séquences	393
13.3.1	Les trois problèmes des HMM	393
13.3.2	Les HMM et la classification bayésienne	394
13.4	L'évaluation de la probabilité d'observation	395
13.5	Le calcul du chemin optimal: l'algorithme de Viterbi	397
13.6	L'apprentissage	400
13.7	Approfondissements	406
13.8	Applications	407
IV Apprentissage par approximation et interpolation		409
14 L'apprentissage bayésien et son approximation		411
14.1	L'apprentissage bayésien	413
14.1.1	Présentation	413
14.1.2	Un petit retour en arrière	415
14.1.3	L'apprentissage bayésien d'une règle de classification	415
14.1.4	La classification bayésienne est optimale en moyenne...	416
14.1.5	...mais on ne peut que l'approcher.	416
14.1.6	La règle bayésienne et la régression aux moindres carrés	418
14.1.7	La règle bayésienne et la minimisation de l'entropie croisée	419

14.1.8	La règle bayésienne et la longueur minimale de description	420
14.1.9	L'apprentissage bayésien non supervisé	422
14.2	Les méthodes paramétriques	422
14.2.1	L'estimation par maximum de vraisemblance	422
14.2.2	L'estimation des paramètres d'une distribution gaussienne	423
14.2.3	Des hypothèses simplificatrices	427
14.2.4	Les cas non gaussiens et multigaussiens	428
14.2.5	La prédiction bayésienne de la distribution des paramètres	428
14.3	L'apprentissage bayésien non paramétrique	431
14.3.1	Généralités : le problème de l'estimation locale d'une densité	431
14.3.2	Les fonctions noyau et les fenêtres de Parzen	432
14.3.3	Les k -plus proches voisins (k -ppv)	435
14.4	Les méthodes semi paramétriques	445
14.4.1	La discrimination logistique	445
14.4.2	Les mélanges de distributions	447
14.4.3	Le cas des réseaux connexionnistes et des arbres de décision	448
15	La classification non supervisée et la découverte automatique	451
15.1	La classification hiérarchique de données numériques	453
15.1.1	Généralités	453
15.1.2	Un algorithme général de classification hiérarchique	456
15.1.3	L'indice du lien simple	456
15.1.4	L'indice de la distance entre centres de gravité	457
15.1.5	L'indice de Ward	457
15.1.6	L'indice de la vraisemblance du lien	458
15.1.7	Le choix du nombre de classes	458
15.2	La classification non hiérarchique de données numériques	458
15.2.1	La méthode des k -moyennes	458
15.2.2	L'estimation d'une somme pondérée de distributions gaussiennes	460
15.2.3	Un exemple	460
15.3	La classification de données symboliques	462
15.3.1	Les données binaires et catégorielles	462
15.3.2	Les attributs nominaux : la représentation attribut-valeur	463
15.3.3	Les données logiques	464
15.4	La découverte automatique	464
15.4.1	Présentation	464
15.4.2	La découverte de fonctions simples	465
15.4.3	Découverte de lois plus complexes	466
15.4.4	Traitement des données bruitées	467
15.4.5	Découverte de lois avec plus de deux variables	468
15.4.6	Améliorations ultérieures	470
15.5	La découverte non supervisée d'associations complexes d'attributs	470
15.5.1	Les associations d'attributs binaires: définitions	470
15.5.2	L'apprentissage des associations	471
15.5.3	Découverte de suites temporelles dans les données	473
15.6	Le coapprentissage et les mélanges d'exemples supervisés et non supervisés	479
15.6.1	Le cas de deux jeux indépendants d'attributs: le coapprentissage	479

15.6.2 L'utilisation de l'algorithme <i>EM</i>	480
16 L'apprentissage de réflexes par renforcement	483
16.1 Description du problème	485
16.1.1 La modélisation d'un agent en action dans le monde	485
16.1.2 Les notions fondamentales	487
16.1.3 Les problèmes et les grandes approches	490
16.2 Si tout est connu : l'utilité de la fonction d'utilité	491
16.3 L'apprentissage des fonctions d'utilité quand l'environnement est connu	492
16.3.1 L'évaluation d'une politique par propagation locale d'information	493
16.3.2 Un théorème conduisant à l'amélioration de politique	494
16.3.3 Processus itératif d'amélioration de politique	495
16.4 Si rien n'est connu : la méthode de Monte-Carlo	496
16.5 Le meilleur des deux mondes : la méthode des différences temporelles	497
16.5.1 L'évaluation suivant la méthode des différences temporelles	497
16.5.2 L'amélioration de politique avec les différences temporelles	498
16.5.3 SARSA : Une méthode d'amélioration « sur politique »	499
16.5.4 Le <i>Q</i> – <i>learning</i> : Une méthode d'amélioration « hors politique »	499
16.5.5 TD(λ) : les méthodes de différences temporelles à plusieurs pas	500
16.6 La généralisation dans l'apprentissage par renforcement	501
16.6.1 Le problème	501
16.6.2 Généralisation par approximation de la fonction d'utilité	502
16.6.3 Méthodes de généralisation par partition de l'espace	504
16.6.4 Méthodes directes d'apprentissage de politique	506
16.7 Le cas des environnements partiellement observables	506
16.8 Exemples d'application	507
16.8.1 Le TD-Gammon	507
16.8.2 Applications au contrôle et à la robotique	508
16.9 Bilan et perspectives	508
V Approfondissements et annexes techniques	511
17 Approfondissement sur l'analyse de l'induction	513
17.1 L'analyse de l'induction de Vapnik	513
17.1.1 Cas où $ \mathcal{H} = \infty$ et $\mathcal{F} \subseteq \mathcal{H}$	514
17.1.2 Fonction de croissance et dimension de Vapnik-Chervonenkis	515
17.1.3 Le lemme de Sauer : un lemme sauveur	517
17.1.4 L'analyse de Vapnik et Chervonenkis pour des fonctions quelconques	520
17.1.5 Discussion	523
17.2 Les principes inductifs avec contrôle de l'espace des hypothèses	524
17.2.1 La minimisation du risque structurel : <i>SRM</i>	524
17.2.2 La théorie de la régularisation	525
17.2.3 La théorie de l'estimation bayésienne	528
17.3 L'induction par compression d'information	529
17.3.1 Un exemple	530
17.3.2 La théorie de l'induction selon Solomonoff	530

17.3.3 La complexité de Kolmogorov	531
17.3.4 Le principe de longueur de description minimale (<i>MDLP</i>)	532
17.3.5 Analyse: compression et pouvoir inductif	534
17.4 L'induction en débat	536
17.4.1 Le <i>no-free-lunch theorem</i> : toutes les méthodes se valent!?	536
17.4.2 Le <i>no-free-lunch theorem</i> et l'analyse de Vapnik: une contradiction? . .	541
17.5 Discussion sur l'analyse classique. Variantes et perspectives	542
17.5.1 D'autres modèles d'apprentissage	544
17.5.2 D'autres types d'analyses	546
18 Annexes techniques	551
18.1 Exemples de fonctions de perte en induction	551
18.1.1 La reconnaissance de formes ou classification	551
18.1.2 La régression	552
18.1.3 L'estimation de densité	553
18.2 Optimisation par descente de gradient	554
18.3 La rétropropagation du gradient de l'erreur	557
18.3.1 Notations	557
18.3.2 Fonctionnement du système	557
18.3.3 Calcul du gradient	558
18.4 Estimation d'une densité de probabilité en un point	560
18.5 L'estimation des paramètres d'une distribution gaussienne	561
18.6 Pourquoi et comment la règle du PPV converge-t-elle?	562
18.6.1 Pourquoi?	562
18.6.2 Comment?	562
18.7 Le calcul de l'intervalle de confiance de l'estimation de la probabilité d'une règle de classification	563
18.8 Pourquoi la règle de décision bayésienne est-elle optimale?	564
18.9 Apprentissage par estimation-maximisation.	565
18.9.1 Un exemple	565
18.9.2 Application de l'algorithme <i>EM</i> à l'exemple	565
18.9.3 Statistique suffisante	566
18.9.4 Plus généralement	566
18.9.5 Retour sur l'exemple	566
18.9.6 L'apprentissage des paramètres des HMM	568
18.9.7 L'apprentissage des paramètres de distributions multigaussiennes	568
Bibliographie	571
Index	586

Première partie

Les fondements de l'apprentissage

Chapitre 1

De l'apprentissage naturel à l'apprentissage artificiel

1.1 L'apprentissage artificiel

Même les machines ont besoin d'apprendre.

Depuis bientôt un demi-siècle, les chercheurs en intelligence artificielle travaillent à programmer des machines capables d'effectuer des tâches qui requièrent de l'intelligence. Nous citerons *l'aide à la décision*, par exemple l'aide au diagnostic médical ; la *reconnaissance de formes*, par exemple la reconnaissance de la parole ou la vision artificielle; le *contrôle de processus*, par exemple la conduite de procédés industriels ; la *prédiction*, par exemple la prédiction de consommation électrique ou la prédiction de cours boursiers ; la *conduite de robots*, y compris d'équipes de robots comme dans la RoboCup¹ ; l'exploration de grandes bases de données (on dit aussi la *fouille de données*), tant il est vrai que si nous croulons sous les informations, il nous manque souvent la connaissance. Chacune de ces tâches et bien d'autres ont stimulé l'inventivité des chercheurs et donné lieu à de nombreuses réalisations impressionnantes. Cependant, programmer des machines capables de s'adapter à toutes les situations et éventuellement d'évoluer en fonction de nouvelles contraintes est difficile. L'enjeu est de contourner cette difficulté en dotant la machine de capacités d'apprentissage lui permettant de tirer parti de son expérience. C'est pourquoi parallèlement aux recherches sur le raisonnement automatique se sont développées des recherches sur l'apprentissage par les machines. Avant cependant d'aborder ce type d'apprentissage, examinons rapidement certaines activités d'apprentissage par des organismes naturels.

L'apprentissage naturel

Dès sa naissance, un enfant apprend à reconnaître l'odeur de sa mère, puis sa voix et plus largement l'ambiance du lieu où il vit. Ensuite, il apprend à coordonner ses perceptions, comme sa vue ou son toucher, avec ses mouvements. Par des *essais* gratifiants ou pénalisants, il apprend plus tard à marcher, manifestant une grande capacité à intégrer des signaux différents : la vue, le sens de l'équilibre, la proprioception, la coordination motrice. Il apprend pendant le même temps à segmenter et catégoriser des sons et à les associer à des significations. Il apprend aussi la structure de sa langue maternelle et acquiert simultanément un répertoire organisé de connaissances sur le monde qui l'environne.

1. La RoboCup est une compétition annuelle organisée depuis 1997 entre équipes de robots footballeurs. Il existe plusieurs types de compétitions mettant en jeu soit des agents simulés, soit des robots réels de tailles diverses. Pour plus d'information, voir par exemple <http://www.robocup.org/>.

Il va aussi apprendre à lire. Il sait déjà faire la distinction entre texte et non texte, parce qu'il a souvent manipulé des livres illustrés où il a observé l'association des images et des symboles de l'écriture. Il apprend d'abord *par cœur* des mots associés à des sons et à leur signification. Plus tard, il extrait des *règles* permettant de distinguer des groupements syllabiques à l'intérieur des mots et de les prononcer. Cet apprentissage est long et *progressif*, il demande des répétitions et des séquences d'exercices bien choisies. Il est en partie *supervisé* par des adultes qui préparent les tâches d'apprentissage, accompagnent son cheminement et sanctionnent, par récompense ou punition, les résultats observés.

Au cours des années qui suivent, l'enfant apprend par étapes à maîtriser des concepts et des opérations de plus en plus abstraits. Finalement, cette fois sans professeur pour l'escorter, il découvrira et énoncera des points de vue personnels, des *théories* sur les phénomènes sociaux, sportifs, économiques, naturels et autres.

Les modalités de l'apprentissage naturel sont donc *multiples*: apprentissage par cœur, par instruction, par généralisation, par découverte, apprentissage impliquant des catégorisations voire la formation de théories, apprentissage plus ou moins supervisé ou autonome, etc. Ces diverses formes d'apprentissage auront-elles une contrepartie lorsqu'il s'agira d'apprentissage par des machines? Et comment envisagera-t-on l'apprentissage naturel après cette étude?

Apprentissage « artificiel » ou apprentissage « automatique »?

Au fait, comment appeler cette discipline? Le terme académique le plus courant est *apprentissage automatique*. Cependant, bien que consacré par l'habitude, ce terme ne nous semble pas complètement satisfaisant. Il sous-entend en particulier une sorte d'activité inconsciente de bas-niveau, qui s'exécute « en tâche de fond » comme disent les informaticiens pour parler d'un processus se déroulant au second plan sans perturber la tâche principale courante. Si certains types d'apprentissages, comme l'habituation, voire même certaines formes d'associations (comme chez le fameux chien de Pavlov), peuvent correspondre à ce schéma, celui-ci est cependant beaucoup trop restrictif.

On peut aussi penser à utiliser les expressions *apprentissage machine* pour traduire directement l'expression américaine *machine learning* ou à *apprentissage algorithmique* pour insister sur les aspects opérationnels.

Il nous semble que la notion d'*apprentissage artificiel* apporte quelque chose de plus profond que la simple idée d'« automatique ». Il est vrai que le mot *artificiel* évoque aussi quelque chose de factice, voire de frelaté et que nous savons combien le terme d'*intelligence artificielle* a souffert de ces connotations; mais nous nous plaçons ici sous le patronage de Herbert Simon (1916-2001), Prix Nobel d'économie et l'un des fondateurs de l'intelligence artificielle, qui a bien su montrer la marque et l'intérêt de la notion de *sciences de l'artificiel* [Sim81].

Sciences naturelles et sciences de l'artificiel

Le projet des *sciences naturelles* est de comprendre les phénomènes en formulant des lois sous-jacentes, de préférence simples. L'ambition fondamentale des *sciences de l'artificiel* n'est pas différente mais, par le but poursuivi et les moyens utilisés, elles s'en écartent cependant suffisamment pour se définir à part entière. Ainsi, le but des sciences de l'artificiel, en particulier de l'apprentissage artificiel, est bien de comprendre les phénomènes de la nature. Mais cette compréhension doit passer par la construction de *modèles* qui (naturellement pour des informaticiens) doivent être capables de réaliser des *simulations*.

Selon le point de vue des sciences de l'artificiel, comprendre implique la capacité de fabriquer pour *reproduire*. Connaître, dans cette optique, c'est concevoir un modèle opératoire du monde

pour le soumettre à des manipulations réglées. Connaître, c'est donc prendre de la distance par rapport à l'objet et se donner les moyens de l'approcher dans son comportement, d'en faire varier des paramètres et d'énoncer des conditions de réalisabilité.

Les sciences de l'artificial présentent deux aspects qui les distinguent des sciences naturelles.

- D'une part, elles conçoivent la connaissance et la compréhension comme une *capacité de simulation*, ce qui implique la possibilité d'explorer *effectivement* les conséquences de postulats initiaux.
- D'autre part, ce sont des sciences qui cherchent des normes permettant de définir ce qu'est un raisonnement valide, un apprentissage correct et les conditions nécessaires pour qu'il puisse avoir lieu. En ceci, les sciences de l'artificial sont aussi des *sciences normatives*, par opposition à l'aspect principalement descriptif des sciences naturelles.

C'est dans ce double sens que nous désirons présenter l'apprentissage artificiel dans cet ouvrage. Certes, il sera bien question d'apprentissage automatisable, donc d'apprentissage automatique et d'un apprentissage réalisable sur des machines, donc d'apprentissage machine, mais l'un des soucis sous-jacents sera de rechercher les conditions de réalisabilité des modèles de l'apprentissage, c'est-à-dire les lois profondes qui règlent la possibilité d'apprendre. Ainsi, l'apprentissage artificiel est la science qui cherche et établit des liens entre les principes généraux d'*apprenabilité* et les méthodes et outils permettant de *réaliser* un apprentissage dans un contexte particulier. La première partie de l'ouvrage est davantage tournée vers l'exposé des principes tandis que le reste présente des techniques justifiées en particulier à la lumière des principes fondamentaux. Le théoricien et l'ingénieur établissent ainsi un dialogue. Nous avons cherché à conserver cet esprit dans l'organisation de l'ouvrage.

1.2 Deux exemples : apprendre à jouer, apprendre à lire

Avant de fixer un cadre méthodologique et de présenter les concepts de base sur lesquels s'organise cet ouvrage, examinons brièvement deux tâches d'apprentissage, simples en apparence, pour faire émerger les multiples questions sous-jacentes.

1.2.1 Apprendre à jouer

Prenons d'abord le cas d'un jeu à deux adversaires sans hasard et sans information cachée. Le jeu d'échecs, le jeu de go ou le morpion en sont les exemples les plus immédiats. Supposons que l'on veuille faire apprendre à une machine à jouer à l'un de ces jeux. Comment s'y prendre ?

Il faut d'abord définir exactement le but poursuivi. S'agit-il de faire apprendre les *règles du jeu* à la machine à partir d'observations de parties jouées ? S'agit-il de lui faire apprendre à *bien jouer* ? S'agit-il de lui faire découvrir les variables pertinentes pour *prédirer* comment jouera l'adversaire (son niveau, son style de jeu) ? S'agit-il de prédire le gagnant, ou bien le nombre de coups restant à jouer ? La liste est déjà variée, elle n'est pas exhaustive.

Prenons maintenant la situation du jeu d'échecs schématisée dans la figure 1.1. Supposons que ce soit à la machine (pièces blanches) de choisir son prochain coup. En supposant qu'elle connaisse les règles du jeu, elle a le choix entre plusieurs dizaines de coups légaux. Lequel est le meilleur ? Pour simplifier, faisons ici le choix que le critère de succès est lié simplement au gain de la partie, en ignorant la durée de jeu, le nombre de coups restant, etc. Comment déterminer le coup à jouer ? L'approche classique en intelligence artificielle utilise l'algorithme MinMax [RN95] fondé sur la notion de *fonction d'évaluation*. En pratique, dans cet algorithme, la machine effectue une recherche en avant dans l'arbre des coups possibles, aussi loin que le temps et les ressources calcul le lui permettent (aux échecs, une dizaine de demi-coups environ). Ensuite, elle *évalue*

chaque position atteinte en fonction de certains critères (par exemple : l'occupation du centre, l'avantage matériel, etc.), et finalement joue le coup lui permettant de maximiser le gain que l'adversaire est obligé de lui concéder. Dans ce cadre, l'apprentissage consiste naturellement à apprendre cette fonction d'évaluation, car c'est elle qui détermine la qualité des décisions.



FIG. 1.1 – Une position dans une partie d'échecs.

Mais d'autres possibilités sont envisageables. Par exemple, la machine pourrait simplement apprendre par cœur une table d'association entre une position et le coup à jouer, une *look-up table* géante. Évidemment, dans le cas des jeux intéressants, ceux qui comportent un très grand espace de situations possibles, cela semble absurde. Mais nous verrons cependant au chapitre 16, dédié à l'apprentissage par renforcement, comment rendre cette idée possible. D'autre part, il est clair que les joueurs humains prennent des décisions motivées par des considérations de stratégie et de tactique : ils opèrent rarement par une exploration exhaustive avec mise en œuvre d'une fonction d'évaluation « simple ». Une simulation plus fidèle de l'apprentissage naturel pourrait chercher à identifier tactiques et stratégies, et à les apprendre directement.

Un problème qui n'a pas encore été abordé est celui du *choix* des données d'apprentissage : quelles sont les informations dont le système apprenant va bénéficier pour apprendre ? Dans le cas du jeu d'échecs, il peut s'agir d'observations de parties. Doit-on lui proposer des parties jouées par des champions, ou des parties médiocres feraient-elles aussi bien l'affaire, ou peut-être des parties jouées par l'apprenant contre lui-même ? Des exemples de coups faibles sont-ils favorables, voire indispensables, à l'apprentissage ? On peut aussi penser profiter de corpus de parties commentées par un professeur, comme dans la littérature échiquierne. Dans le cas de l'apprentissage des règles du jeu, des exemples de coups illégaux seraient-ils favorables, voire indispensables, à l'apprentissage ?

Il se pose aussi la question du *séquencement* des leçons : y a-t-il un ordre de présentation plus favorable qu'un autre ? Doit-on tenir compte d'une vitesse d'assimilation, comme chez les apprenants humains, liée aux capacités computationnelles de l'apprenant ?

Autant de questions qui ressortent tant d'une analyse théorique sur les conditions de possibilité de l'apprentissage que d'une étude expérimentale.

Finalement on n'évitera pas le problème de la *validation* de l'apprentissage réalisé. Comment mesurer la performance de l'apprenant après l'apprentissage ? En comptant la proportion de parties gagnées contre un échantillon représentatif de joueurs ? En incluant le temps de réflexion ? En demandant des explications des décisions prises ? Et dans ce cas, qui jugera de leur validité ?

Même dans le contexte familier et apparemment simple de l'apprentissage d'un jeu, on voit donc que l'ensemble des questions ouvertes est vaste.

1.2.2 Apprendre à reconnaître des caractères manuscrits

Maintenant, supposons que nous voulions entraîner une machine à reconnaître des caractères manuscrits tels qu'ils apparaissent sur une enveloppe timbrée, c'est-à-dire en général assez bien tracés et séparés les uns des autres. La figure 1.2 donne un exemple de caractères tels qu'ils peuvent se présenter dans la réalité. Comment une machine peut-elle apprendre à identifier ces formes ?

Le codage

La difficulté de base est que la variété des formes rencontrées est infinie. Il ne peut donc être question d'apprentissage par cœur. Il faut par conséquent, à partir d'un échantillon d'exemples « bien choisis »(comment ?) être capable de *généraliser*. De manière informelle, nous définissons un exemple comme l'association d'une forme et d'une étiquette. C'est ainsi que la forme de la figure 1.2 est associée à l'étiquette 'a' (lettre de la catégorie 'a'). Nous avons ici affaire à ce qu'on appelle de l'*apprentissage supervisé*².

Ici se pose la première question : comment coder les formes ? Par une matrice binaire transcrivant l'éclairement des pixels de la rétine de la caméra ? Dans cette hypothèse, chaque caractère serait défini par une matrice, disons de taille 16×32 ³. Avant de s'intéresser à l'exploitation de telles représentations, il faut résoudre un problème d'homogénéité. Les caractères seront-ils centrés sur la rétine ? Seront-ils tous à la même échelle ? Auront-ils une orientation imposée ? On voit que même dans le cas d'un codage très primitif des formes, un *prétraitement* est indispensable.

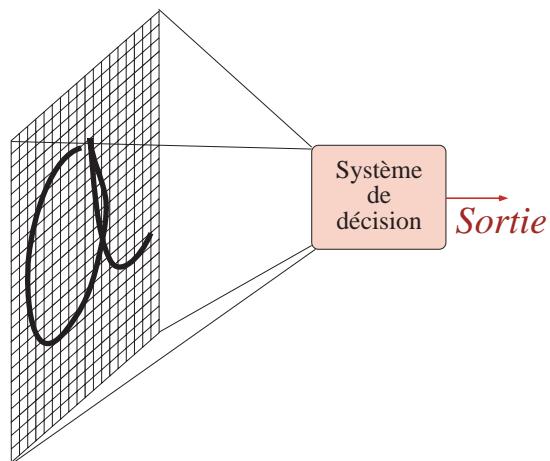


FIG. 1.2 – Une tâche de reconnaissance de caractères manuscrits.

Pourquoi ne pas d'emblée coder les formes de manière plus pertinente, en extrayant des caractéristiques essentielles ou invariantes ? Par exemple la présence de boucles, ou bien la hauteur ou la largeur ou le nombre de points de retour. Il est probable que l'apprentissage subséquent s'en trouverait facilité. Mais... qui a appris ce codage ? Et comment ?

2. Par contraste avec l'apprentissage non supervisé dans lequel les étiquettes ne sont pas fournies.
3. Notons qu'après ce codage le nombre de formes possible n'est plus *stricto sensu* infini. Si chaque pixel est noir ou blanc, le nombre de formes différentes est cependant de $2^{16 \times 32} \simeq 10^{30}$.

Alors que la description des formes comme des projections sur la rétine de la caméra est immédiate, une redescription adaptée à l'apprentissage implique des opérations non triviales et surtout des connaissances *a priori* sur le mécanisme de cet apprentissage. Il s'agit d'éliminer les descripteurs non pertinents, par exemple la couleur de l'encre ou celle du fond de l'image, de recoder pour tenir compte des invariances par translation ou par changement d'échelle, voire d'introduire de nouveaux descripteurs. Certains de ces nouveaux descripteurs, non présents dans la description brute des données, n'impliquent pas nécessairement des attributs complexes. Ainsi, pour distinguer un '*a*' d'un '*b*', il suffit en général de considérer le rapport de leur hauteur à leur largeur. Mais le plus souvent, il faudra être capable d'inventer des descripteurs sophistiqués. Une autre technique consiste à introduire une grande collection de descripteurs dont l'apprentissage essaiera de tirer le meilleur parti.

La mesure de performance

Retournons maintenant au problème de la définition du critère de performance. S'agit-il simplement du nombre d'erreurs de classification après apprentissage, que l'on peut ramener à une *probabilité de mauvaise classification*? S'agit-il d'une mesure de risque plus élaborée, prenant en compte le fait qu'il vaut mieux se tromper sur une lettre que sur un chiffre (le code postal est plus difficile à reconstruire que le nom de la commune)? Ici encore, les possibilités sont nombreuses et c'est l'application qui commande le choix à faire.

Dans tous les cas, l'évaluation de l'apprentissage devra être conduite avec soin. En général, on mesure la performance après que l'apprentissage a eu lieu sur un certain nombre de données que l'on appelle *échantillon d'apprentissage*. Si l'on fait varier la taille de cet échantillon, on obtient une *courbe d'apprentissage* comme celle donnée sur la figure 1.3. Cependant, il faut s'assurer que la mesure de performance s'effectue sur un *échantillon de test* différent de l'échantillon d'apprentissage. Autrement, ce ne serait pas la capacité de généralisation qui serait testée, mais une capacité à l'*apprentissage par cœur*, qui n'est pas pertinente dans ce contexte (mais qui pourrait éventuellement l'être dans le cas de caractères d'imprimerie).

La modélisation

Finalement, il faudra décider de la forme d'apprentissage à réaliser, c'est-à-dire de ce qui est appris en interne par le système apprenant. Pour donner un exemple, on peut se figurer les caractères comme étant décrits dans un espace de descripteurs à plusieurs dimensions. Certains des points de cet espace correspondent à la lettre '*a*', d'autres à la lettre '*b*', etc. Le problème est alors d'apprendre à associer à chaque point la lettre correspondante. Ceci peut se faire de plusieurs manières. Le but de cet ouvrage est de les présenter et d'offrir un cadre conceptuel pour orienter les choix à faire. Pour donner déjà quelques exemples, on peut imaginer une approche *géométrique*: apprendre des frontières entre les régions correspondant aux différentes classes. Une nouvelle forme inconnue sera alors étiquetée en fonction de sa place par rapport aux frontières trouvées. On peut aussi adopter un point de vue *probabiliste* et apprendre des probabilités conditionnelles d'appartenance des points aux classes, ou réciproquement des probabilités conditionnelles des classes connaissant la description des points. On pourrait également envisager d'utiliser un critère de décision par les *plus proches voisins* dont on connaît l'étiquette. Il faudra alors disposer d'un nombre suffisant de points étiquetés et d'une relation de voisinage définie proprement. Et il existe encore bien d'autres possibilités...

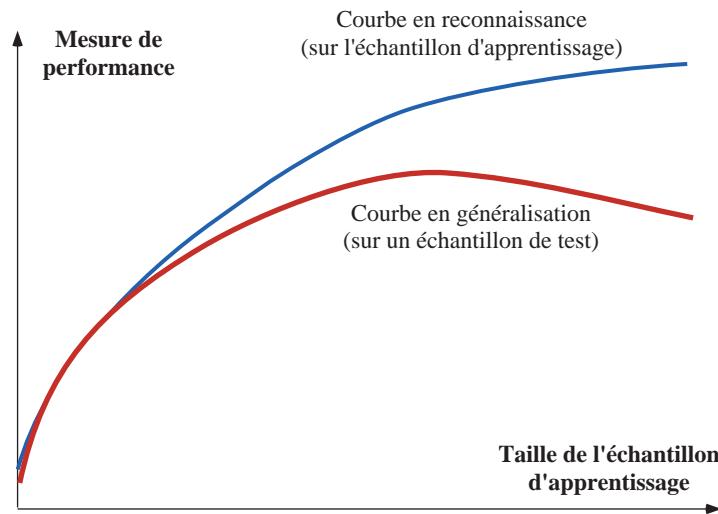


FIG. 1.3 – Une courbe de performance classique. On suppose ici que l'apprentissage est supervisé : la machine peut exploiter les informations contenues dans un échantillon d'apprentissage en vue de faire des prédictions sur des observations nouvelles. En abscisse figure l'exploitation de l'échantillon d'apprentissage, par exemple mesurée en nombre d'examens de la base d'apprentissage. En ordonnée est portée une mesure de l'erreur, soit l'erreur faite par le système en prédiction sur des exemples vus en apprentissage (erreur de reconnaissance), soit sur des exemples non vus (erreur en généralisation). Normalement, la performance s'améliore au fur et à mesure que l'algorithme exploite davantage l'information. Elle est généralement meilleure en reconnaissance qu'en généralisation. On observe aussi fréquemment une baisse des performances en généralisation lorsque l'apprenant exploite « trop » les données. Ce phénomène qui peut sembler paradoxal trouvera son explication dans la suite de l'ouvrage.

1.3 Deux approches : la cybernétique et les sciences cognitives

Pour commencer à réfléchir à l'apprentissage, il faut d'abord évoquer d'autres notions. D'abord, celles d'*évolution*, de *transformation*, de *modification*. Il n'y a pas d'apprentissage sans évolution. Mais cette notion d'évolution n'est pas suffisante. Le vin aussi évolue, on ne dira pourtant pas qu'il apprend. Il faut donc y ajouter la notion de *progrès*. L'apprentissage vise à rendre plus performant, meilleur, qu'elle qu'en soit la mesure. Mais le vin qui se bonifie apprend-il ? Non. C'est que l'apprentissage implique aussi une notion d'*adaptation* à un environnement, à une tâche. Il faut qu'il y ait une sorte de rétroaction de l'environnement sur le système pour que l'on puisse qualifier d'apprentissage le processus de transformation qui en découle éventuellement. Si le vin réagit à son environnement, c'est de manière minimale, totalement passive, comme tout objet ou matière. Finalement, l'apprentissage évoque aussi souvent la notion de *transfert* à d'autres situations, voire de *généralisation*. Quelqu'un qui apprend par cœur n'est qu'un perroquet ou un « savant idiot ». On attend plus d'un authentique apprentissage, à savoir qu'il y ait capacité à savoir tirer profit d'une expérience particulière pour faire face à une nouvelle situation suffisamment « proche » pour que le transfert de l'une à l'autre puisse être pertinent.

Transformation, progrès, adaptation, transfert et généralisation : chacune de ces notions ren-

voie à son tour à des questions que toute étude de l'apprentissage devra aborder.

- Qu'est-ce qui est transformé lors de l'apprentissage ? En particulier, dans une machine apprenante, comment représente-t-on ce qui détermine le comportement du système et qui subira éventuellement une modification ? Le neurobiologiste évoquera immédiatement le support biologique : les neurones, leurs connexions, les neurotransmetteurs ; le psychologue raisonnera en termes de croyances, de connaissances ; l'informaticien envisagera des réseaux de neurones artificiels, des assertions Prolog d'un système expert, etc.
- Comment peut s'effectuer le processus de transformation ?
- En réponse à quel type de sollicitation ? Qu'est-ce qui informe le système apprenant qu'il doit se transformer ? Qu'est-ce qui lui permet de mesurer son progrès ou son degré d'adaptation ?
- Comment un observateur extérieur peut-il mesurer le progrès et attribuer à un système une capacité d'apprentissage et une amélioration de performance ?
- Comment caractériser le transfert entre deux situations ? Comment mesurer leurs différences et ce que l'expérience de l'une apporte à l'expérience de l'autre ?

Nous n'épuisons pas ici l'ensemble des questions qui peuvent se poser à propos de l'apprentissage. Nous voulons seulement signaler les plus évidentes, les plus immédiates. Nous l'avons fait dans des termes volontairement généraux et relativement vagues pour ne pas d'emblée embrasser un point de vue, un type de questionnement. Dans cette section, sans vouloir, ni pouvoir naturellement être exhaustif, nous voulons brosser à grands traits deux modes d'approche qui ont constitué deux moments de l'étude de la cognition au XX^e siècle, et qui ont façonné le point de vue de l'apprentissage artificiel durant ces dernières décennies.

1.3.1 La cybernétique

Dans les années quarante et cinquante, certains scientifiques se donnent pour ambition d'édifier une science générale du fonctionnement de l'esprit. Sous l'influence du mathématicien Norbert Wiener (1894-1964), ils identifient ce programme par le terme de *cybernétique*. Leurs travaux, leurs discussions et débats donnèrent lieu à dix conférences tenues à New-York sous le nom de conférences Macy du nom de la fondation philanthropique les finançant. Une partie des papiers fondateurs de ce mouvement sont reproduits dans [PT95], tandis que [Dup94] retrace l'historique et les clivages conceptuels de la cybernétique et des sciences cognitives orthodoxes.

L'un des postulats des cybernéticiens de la première heure, c'est qu'il est vain de vouloir appréhender directement les notions d'intelligence, de conscience, de mémoire, d'anticipation, d'intentionnalité. Pour étudier l'esprit, il faut le *naturaliser* ou le *matérialiser* et pour cela, d'une part *assimiler l'esprit à l'activité du cerveau* et d'autre part, *poser que celui-ci est une machine*. Il en découle une triple focalisation, d'une part sur l'agent cognitif considéré individuellement (plutôt que par exemple sur le fonctionnement global de la culture, de la société ou de l'environnement), d'autre part sur les mécanismes matériels, par exemple neurobiologiques, agissant à l'intérieur de la machine, et enfin sur les règles logiques qui sous-tendent le comportement mesuré.

Un *agent cognitif* devient ainsi un module opérant sur des *entrées* pour les transformer en *sorties*. La notion de sujet disparaît pour faire place aux concepts de programme, syntaxe et information telles qu'elles figurent dans la définition de la machine de Turing (1912-1954), des systèmes asservis et autorégulés de Wiener, et de la théorie de l'information de Shannon (1916-2001). On cherche donc à préciser les entrées dont dispose un système, en essayant d'évaluer la quantité d'informations disponibles et corollairement l'effet d'une quantité plus ou moins grande

d'informations sur la performance de l'agent. Les cybernéticiens se concentrent également sur les modules fonctionnels qui permettent de réaliser une fonction supérieure donnée, telle que la capacité à s'orienter dans l'espace ou à prendre une décision. De même, on cherche de quels comportements génériques est capable une structure donnée de modules fonctionnels. Plus tard, lors de la « deuxième cybernétique », on étendra cette quête à l'étude des totalités engendrées par une collection de relations et aux systèmes auto-organisés.

Dans ce cadre, les questions concernant l'apprentissage deviennent relatives à l'effet d'une quantité croissante d'informations sur la performance du système : comment celui-ci s'adapte à de nouvelles stimulations ou à un nouvel environnement, comment il retrouve un nouvel équilibre. On cherche également ce qu'une modification de structure peut entraîner comme changement de performance. Avec la deuxième cybernétique, l'apprentissage devient complètement lié à la capacité du système à s'autoadapter ou à continuer à « exister » dans une nouvelle situation. Le cybernéticien se préoccupe de définir une structure d'unités en interaction et d'étudier, par simulation, ses capacités d'adaptation spontanée à toutes sortes de milieux.

Ce qui est intéressant, c'est que la dimension dynamique de l'apprentissage et sa fonction d'adaptation à un milieu sont prises en compte. En revanche, la mise en valeur trop exclusive de simulations expérimentales et de reproductions de comportements, par exemple éthologiques, a jusqu'à présent nuit à une construction théorique de l'apprentissage. Il y a pléthore d'expériences singulières, parfois spectaculaires, et pénurie de cadres théoriques.

Le cognitivisme, qui est en partie héritier de la cybernétique, offre curieusement une image presque inversée, nous allons le voir, avec une myopie sur l'aspect dynamique de l'apprentissage et son rôle adaptatif, mais avec une forte construction théorique.

1.3.2 Le pari du cognitivisme

Le cognitivisme reprend à son compte l'*approche fonctionnaliste de l'esprit* selon laquelle celui-ci peut, d'une part, être abordé par ses manifestations sans avoir à présupposer de quelques facultés mentalistes, et, d'autre part, être considéré comme *une fonction calculable*, c'est-à-dire réalisable par une machine de Turing. Cette dernière idée permet en outre d'envisager la séparation de l'aspect matériel (le *hardware*) de la machine de son aspect logiciel. De cette manière, c'est entièrement sur le logiciel, les programmes, que se trouve reportée la charge de toutes les éventuelles propriétés cognitives du système.

Mais le cognitivisme va plus loin. Si pour lui, comme pour la cybernétique, penser c'est *calculer* comme un ordinateur, il ne s'agit pas en revanche de manipuler des symboles dénués de sens, mais de manipulation réglée de *symboles ayant le statut de représentation d'état du monde*. Pour le dire autrement, les symboles envisagés par le cognitivisme ont à la fois une réalité matérielle et une valeur sémantique. Ils représentent certains aspects du monde, et, de ce fait, les calculs opérés par la machine deviennent une simulation qui préserve la structure de la réalité. À toute opération « mentale » correspond ainsi une transformation possible du monde. Le cognitivisme pose de la sorte des contraintes beaucoup plus fortes sur le système de symboles manipulés que la cybernétique. Pour donner un exemple, les nombres manipulés dans les réseaux connexionnistes n'ont pas le statut de symboles pour le cognitivisme, et si un cognitiviste étudie un tel réseau, il le fera en l'abordant à un autre niveau, dans lequel il pourra lui attribuer des connaissances et des règles de raisonnement.

Il s'agit alors d'énoncer comment est constituée une représentation et ce qui peut rendre sa manipulation sémantiquement correcte. Pour ce faire, le cognitivisme s'est naturellement trouvé influencé par le mouvement de pensée le plus caractéristique et le plus influent du XX^e siècle, à savoir celui selon lequel la réalité est organisée comme un langage. À l'instar d'autres disci-

plines comme la biologie, maintenant entièrement conçue comme élucidation du code génétique, l'ethnologie de Levi-Strauss inscrite dans le mouvement structuraliste, la psychanalyse cherchant le code de l'inconscient, et même la physique⁴, les sciences cognitives ont été chercher du côté de la philosophie analytique – essentiellement une philosophie du langage – une solution à leur problème. Selon cette approche, la pensée procède à partir de propositions portant sur le monde, dotées d'une syntaxe, et manipulées suivant des *règles d'inférence* strictes d'un langage formel, parmi lesquelles figurent au premier plan la déduction, l'abduction, la généralisation, etc., c'est-à-dire des règles d'inférence liées à la logique.

On ne s'attardera pas ici sur les difficultés de nature philosophique rencontrées par cette approche, touchant entre autres au problème de la référence et de l'intentionnalité (voir par exemple l'excellent livre de Joëlle Proust [Pro97]). En revanche, il est important de souligner les conséquences de ce point de vue pour l'étude de l'apprentissage.

À partir du moment où la cognition est considérée comme la manipulation, suivant un ensemble de règles strictes, de propositions sur le monde, il devient naturel d'envisager l'ensemble de toutes les propositions possibles sur le monde et de toutes les théories correspondantes. Cet ensemble de « mondes possibles » sera assimilé à un espace d'hypothèses potentielles pouvant expliquer les manifestations observées du monde. Selon cette perspective, l'apprentissage devient la recherche d'une, ou plusieurs, hypothèse(s), s'accordant aux données recueillies jusque-là. Et l'étude de l'apprentissage selon ce point de vue amène alors naturellement à considérer un certain nombre de questions.

Nous invitons ici le lecteur à faire une pause et à s'interroger sur le programme de recherche qu'il envisagerait à ce point. Il s'agit là en effet de tester à quel point un parti pris philosophique sur un sujet détermine ensuite de manière très profonde la nature du questionnement qui va façonner toute la discipline.

Voici, par exemple, un ensemble de questions fondamentales qui façonnent le point de vue cognitiviste :

- Comment évolue l'espace d'hypothèses en fonction des données disponibles sur le monde ? Peut-on montrer qu'il se réduit ? Si oui, à quelle vitesse ? Peut-il y avoir espoir de converger vers une hypothèse unique ? Et si, après qu'un certain nombre d'éléments d'informations a été recueilli, il reste plusieurs hypothèses, peut-il y avoir espoir qu'elles aient une parenté ou un proximité pour pouvoir, sans trop de risque, en choisir une plutôt qu'une autre ?
- Comment peut-on rendre efficace l'exploration de l'espace des hypothèses en cours d'apprentissage ?
- Quelles sont les règles d'inférence appropriées pour rendre compte de l'apprentissage ? En particulier, par quels opérateurs peut-on modéliser les règles d'inférence telles que l'induction, l'analogie, l'abduction, qui correspondent à des formes de raisonnement permettant d'élaborer une représentation plus opératoire que la simple mémorisation des entrées ?

À ce stade, ces questions sont encore vagues. Nous verrons comment elles peuvent être précisées dans les modèles et les algorithmes de l'apprentissage, comme dans le cadre des théories de l'apprenabilité. Il ne faudra pour autant pas oublier le formidable réductionnisme dont elles sont issues, en particulier la projection opérée ainsi de la cognition sur le plan des représentations et des règles de la logique formelle. Par ailleurs, à trop se mettre au niveau de la proposition

4. Dont une grande partie de l'activité est tournée vers l'exploration des conséquences de manipulations du langage mathématique (par exemple la théorie des groupes qui permet de prédire, et avec quel succès, l'existence de particules). Si la physique s'enivre d'une telle puissance, elle ne peut que s'interroger sur « le pouvoir prédictif déraisonnable des mathématiques » [Wig60].

et de la règle d'inférence, la scène globale, le fonctionnement des théories et de la connaissance en général finissent par être occultés. C'est sans doute le prix à payer pour faire les premiers pas. Il ne faut cependant pas en devenir dupe et négliger l'objectif qui est la compréhension de l'apprentissage, en particulier dans sa dimension liée à la construction de connaissances, telle que l'envisage par exemple la didactique.

Nous pensons que le reste de l'ouvrage montrera que l'apprentissage artificiel tel qu'il est abordé actuellement est placé à la résultante des influences de la cybernétique et du cognitivisme. Du second, il a pris les concepts d'espace d'hypothèses, de langage de représentation, de règles d'inférence, de recherche dans un espace de possibilités. Cela a rendu possible les premières simulations, les premiers programmes. Mais il a fallu aussi l'apport du premier courant pour que, en relâchant les contraintes sur les symboles manipulés, on ose se servir de toute une panoplie d'outils mathématiques permettant de caractériser l'apprentissage comme un processus de convergence vers une fonction cible. Il reste sans doute maintenant à dépasser ces deux perspectives, mais pour cela il est utile de les connaître et d'en peser l'influence.

La suite de ce chapitre vise à fournir les concepts et notations de base nécessaires à la compréhension du reste de l'ouvrage. Les lecteurs intéressés par un débat entre plusieurs perspectives sur l'apprentissage (incluant la didactique, la psychologie et l'informatique) peuvent se reporter par exemple à [TNC^{+re}].

1.4 Les concepts de base de l'apprentissage

Afin de présenter les principaux concepts permettant d'étudier l'apprentissage, nous allons nous référer dans ce chapitre introductif à un scénario certes limité, mais servant de cadre idéal à la majorité des recherches actuelles sur l'apprentissage artificiel : celui de l'induction supervisée. Il faudra bien sûr l'aménager, voire le bousculer, à l'occasion, mais il fournit un bon point de départ.

Qu'est-ce que l'induction ? C'est le processus par lequel on tire des lois de portée générale en partant de l'observation de cas particuliers. C'est ainsi que l'avimateur novice évoqué dans la préface cherche une loi générale lui permettant de distinguer les oies des cygnes. Pour ce faire, il a à sa disposition quelques exemples de volatiles, sur lesquels il effectue des mesures (la couleur, la taille, par exemple). De plus, il dispose d'un expert qui lui dit à quelle espèce appartient chacun d'eux. Dans ce cas, dans lequel un « oracle » fournit la bonne réponse, on parle d'induction supervisée. Nous définissons ceci plus formellement ci-dessous.

1.4.1 Un scénario de base pour l'induction

Dans ce scénario, nous supposons qu'un *système apprenant* reçoit des données de l'univers dans lequel il est placé. Nous ne nous interrogerons pas ici sur ce qui permet de définir et de segmenter ces « données ». Dans le cas de l'apprentissage supervisé, chacune de ces données prend la forme d'un couple dans lequel on distingue d'une part la description d'une situation ou encore *observation* (par exemple une situation de jeu, ou bien une matrice de niveaux de gris), et d'autre part une réponse, que l'on appelle aussi fréquemment *sortie désirée*, (par exemple *situation de mat en trois coups* ou bien *lettre 'a'*, ou encore *volatile = cygne*) qui est supposée être fournie par un *oracle*. Une donnée \mathbf{z}_i est ainsi définie par un couple (*observation*, *sortie désirée*) que nous noterons $(\mathbf{x}_i, \mathbf{u}_i)$ ⁵. Voir la figure 1.5.

5. Nous utilisons la lettre \mathbf{u} en accord avec les notations de l'automatique et de la théorie du contrôle, nous réservant la lettre \mathbf{y}_i pour indiquer la sortie produite par le système apprenant en réponse à l'entrée \mathbf{x}_i .

1.4.2 Quelques notions clés

1.4.2.1 Le critère de succès

Dans le scénario de l'apprentissage supervisé, la tâche de l'apprenant est d'essayer d'approcher au mieux la sortie désirée \mathbf{u}_i pour chaque entrée observée \mathbf{x}_i . Dans le cas idéal, l'apprenant devient capable, après un certain temps d'apprentissage, de prédire exactement, pour chaque entrée \mathbf{x}_i , la sortie désirée \mathbf{u}_i . En général cependant, il faudra se contenter d'une approximation de la réponse de l'oracle. Nous formaliserons plus loin cette notion d'approximation grâce à une *fonction de risque*, encore appelée *critère de succès*, qui dépendra à la fois du domaine étudié et de l'objectif de l'apprentissage.

Le critère de succès est ce qui est mesuré dans l'évaluation de la performance. Il s'agit donc d'un *critère relatif à un observateur externe*. Par exemple, la performance sera mesurée en fonction du nombre d'erreurs commises par l'apprenant en cours d'apprentissage, ou en fonction de son taux d'erreur après apprentissage. Dans une tâche qui prend une importance grandissante avec le réseau Internet, celle qui consiste à chercher des documents relatifs à une requête particulière, la performance sera fonction à la fois du nombre de documents pertinents trouvés par le système rapporté au nombre réel de documents pertinents et du nombre de documents pertinents non trouvés. Plus généralement, la mesure de performance peut inclure des facteurs indépendants de l'adéquation aux données d'apprentissage et de natures très diverses. Par exemple, la *simplicité* du résultat d'apprentissage produit par l'apprenant, sa *compréhensibilité*, son *intelligibilité* par un expert, la facilité de son intégration dans une théorie courante, le faible coût computationnel nécessaire à son obtention, etc.

Il faut ici faire une remarque importante. Le critère de succès, mesuré par un observateur externe, n'est pas nécessairement identique à la *fonction de coût* ou *de perte* qui est *interne à l'apprenant* et le guide pour faire converger les paramètres de son modèle d'apprentissage. Par exemple, un algorithme d'apprentissage de réseau connexioniste cherche généralement à minimiser un écart quadratique entre ce qu'il prédit sur chaque exemple d'apprentissage et la sortie désirée. Cette mesure est interne au système, elle lui permet de mesurer la qualité de l'approximation de son modèle courant avec les données d'apprentissage, mais elle n'est pas généralement celle qui intéresse l'observateur externe qui examine par exemple le taux d'erreur ou prend éventuellement en compte d'autres critères comme ceux évoqués plus haut.

1.4.2.2 Notion de protocole d'apprentissage

L'apprentissage et son évaluation dépendent du *protocole* qui règle les interactions entre l'apprenant et son environnement, incluant l'oracle. Il faut ainsi distinguer l'*apprentissage hors ligne* (*batch learning*), dans lequel toutes les données d'apprentissage sont fournies d'un seul coup à l'apprenant, de l'apprentissage séquentiel, incrémental ou *apprentissage en ligne* (*on-line learning*) dans lequel les données arrivent en séquences et où l'apprenant doit délibérer et fournir une réponse après chaque entrée ou groupe d'entrées.

Le protocole stipule également le type d'entrées fournies à l'apprenant et le type de sorties attendues. Par exemple, un scénario peut spécifier qu'à chaque instant l'apprenant reçoit une observation \mathbf{x}_i , qu'il doit alors fournir une réponse \mathbf{y}_i , et que seulement alors l'oracle produit la réponse correcte \mathbf{u}_i . (Voir figure 1.4, partie gauche). Une illustration d'un tel protocole est fournie par le cas d'un système essayant de prédire le cours du lendemain d'un titre en bourse à partir d'informations sur le contexte économique courant. À chaque fois la réponse correcte est disponible le lendemain seulement. On parle alors naturellement de tâche de *prédiction*. Plus fondamentalement, les tâches dites de prédiction s'intéressent à prévoir correctement une

réponse en un point précis : quel sera le cours du soja demain à midi, la consommation électrique dans trois mois, quelle est la pathologie de ce patient particulier, etc.

Les tâches de prédiction sont à contraster avec les *tâches d'identification* dans lesquelles le but est de trouver une explication globale parmi toutes celles possibles, qui une fois connue permettra de faire des prédictions quelle que soit la question. Un système d'analyse boursière pourrait ainsi chercher à identifier la fonction suivie par le cours du soja. De même, un fournisseur d'électricité pourrait vouloir connaître l'ensemble de la courbe de la consommation sur une année. Dans le cas de la médecine, une tâche d'identification consisterait à trouver des lois permettant de faire un diagnostic pour n'importe quel malade et non pour un patient particulier.

Le scénario sera alors différent. Par exemple, il pourra prévoir que le système apprenant doit fournir après chaque nouvelle entrée ($\mathbf{x}_i, \mathbf{u}_i$) une hypothèse sur la « fonction cachée » de l'oracle par laquelle celui-ci détermine \mathbf{u}_i en fonction de \mathbf{x}_i . On conçoit que le critère de succès ne soit pas le même dans le cas d'une tâche de prédiction que dans celui d'une tâche d'identification. Dans ce dernier cas, en effet, on demande beaucoup plus à l'apprenant puisqu'on attend de lui une hypothèse explicite, donc une sorte d'explication de ses prédictions (voir figure 1.4, partie droite).

Par ailleurs, l'apprenant peut être plus ou moins *actif*. Dans les protocoles décrits jusqu'ici, l'apprenant reçoit passivement les données sans avoir d'influence sur leur sélection. Il est possible d'envisager des scénarios dans lesquels l'apprenant a une certaine initiative dans la recherche d'informations. Dans certains cas, cette initiative est limitée, par exemple lorsque l'apprenant, sans avoir la totale maîtrise du choix de l'échantillon d'apprentissage, est simplement capable d'orienter sa distribution de probabilité. Les méthodes de *boosting*, décrites dans le chapitre 11, en sont une illustration. Dans d'autres cas, l'apprenant peut poser des questions sur la classe d'appartenance d'une observation, on parle alors d'*apprentissage par requête d'appartenance (membership queries)*, ou même organiser des expériences sur le monde, et on parle alors d'*apprentissage actif*. Le jeu de MasterMind, qui consiste à deviner une configuration de pions de couleurs cachés en posant des questions suivant certaines règles, est un exemple simple d'apprentissage actif dans lequel l'apprenant possède l'initiative des questions.

1.4.2.3 Notion de tâche d'apprentissage

Il est possible d'aborder l'objectif du processus d'apprentissage suivant plusieurs points de vue.

Par rapport à la connaissance

Le but de l'apprentissage peut être de *modifier le contenu* de la connaissance⁶. On parle alors d'*acquisition de connaissances*, de *révision*, et, pourquoi pas, d'*oubli*. En parlant de manière informelle, l'apprenant sait désormais plus ou moins de choses. Mais cela ne préjuge pas de sa capacité à utiliser ses connaissances.

Le but de l'apprentissage peut aussi être, sans nécessairement modifier le « contenu » de la connaissance, de *le rendre plus efficace* par rapport à un certain but, par réorganisation, optimisation ou compilation par exemple. Ce pourrait être le cas d'un joueur d'échecs ou d'un calculateur mental qui apprend à aller de plus en plus vite sans pour autant connaître de nouvelles règles de jeu ou de calcul. On parle dans ce cas d'*optimisation de performance (speed-up learning)*.

6. Qui peut-être mesuré par exemple par sa clôture déductive, c'est-à-dire, dans une représentation logique, tout ce qui peut être déduit correctement à partir de la base de connaissances courante.

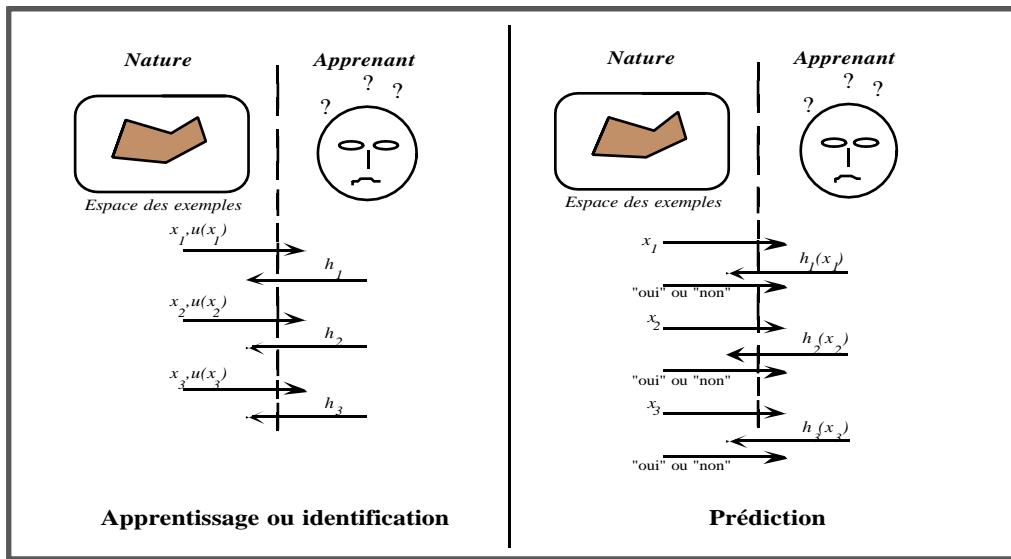


FIG. 1.4 – Différence entre un protocole d'identification (à gauche) et un protocole de prédiction (à droite). Dans le protocole d'identification, l'objectif de l'apprenant est de trouver une hypothèse h qui rende bien compte des données d'apprentissage. Dans le protocole de prédiction, l'apprenant doit « seulement » chercher à trouver la bonne réponse à une question particulière. Ici, on suppose qu'après chaque prédiction, l'apprenant reçoit une confirmation ou une infirmation de sa prédiction.

Par rapport à l'environnement

La tâche de l'apprentissage peut aussi être définie par rapport à ce que l'agent apprenant doit réaliser pour « survivre » dans son environnement. Cela peut inclure :

- *Apprendre à reconnaître* des formes (par exemple : des caractères manuscrits, des oiseaux, des prédateurs, une tendance haussière à la bourse, une appendicite, etc.). Lorsque l'apprentissage se fait avec un professeur, ou oracle, qui fournit les réponses désirées, on parle d'*apprentissage supervisé*. Sinon, on parle d'*apprentissage non supervisé*. Dans ce cas, la tâche d'apprentissage consiste à la fois à découvrir des catégories et à trouver des règles de catégorisation.
- *Apprendre à prédire*. Il y a alors une notion de dépendance temporelle ou de causalité.
- *Apprendre à être plus efficace*. C'est le cas notamment des situations de résolution de problème, ou de recherche de plans d'action dans le monde.

Par rapport à des classes abstraites de problèmes

Indépendamment même d'un algorithme d'apprentissage, il est possible de caractériser l'apprentissage par une classe générale et abstraite de problèmes et de processus de résolution qui leur sont liés. C'est ainsi qu'un certain nombre de disciplines, en particulier issues des mathématiques ou de la théorie de l'information, se sont découvertes un intérêt pour les problèmes d'apprentissage.

- *Les théories de compression d'information*. En un certain sens, l'apprentissage peut être abordé comme un problème d'extraction et de compression d'information. Il s'agit d'extraire l'information essentielle ou le message initial d'un émetteur idéal, débarassé de toutes

ses redondances. En un sens, les sciences d'observation, telles l'astronomie ou l'ornithologie, procèdent par élimination des détails superflus ou redondants et par la mise en évidence de régularités cachées.

- *La cryptographie.* Dans une perspective analogue, proche des préoccupations de la théorie de l'information, l'apprentissage peut être considéré comme une *tâche de décodage* ou même de décryptage d'un message codé par l'émetteur idéal et intercepté en tout ou partie par l'agent. Après tout, c'est parfois ainsi qu'est vu le scientifique étudiant la nature. Il est alors logique de se demander sous quelles conditions un message peut être « cassé », c'est-à-dire sous quelles conditions un apprentissage est possible.
- *L'analyse.* L'apprentissage peut également être examiné comme un *problème d'approximation*. C'est particulièrement clair lorsque l'on adopte le point de vue de la figure 1.5. La tâche de l'apprenant est bien alors de trouver une approximation aussi bonne que possible d'une fonction cachée connue uniquement par l'intermédiaire d'un échantillon de données. Le problème de l'apprentissage devient alors souvent celui de l'étude des conditions d'approximation et de convergence. Nous aurons largement l'occasion de développer ce point de vue, actuellement dominant, dans le chapitre 2.
- *L'induction.* Dans les années soixante-dix et au début des années quatre-vingt, sous l'influence du point de vue cognitiviste, une large communauté de chercheurs, particulièrement active en France, s'est penchée sur l'apprentissage en tant que *problème de généralisation*. Cette approche part de deux présupposés essentiels. D'une part, l'agent cognitif apprenant doit apprendre quelque chose qu'un autre agent cognitif équivalent connaît. Il est donc normalement capable d'atteindre parfaitement la connaissance cible. D'autre part, les connaissances et les données peuvent être décrites par un langage. On cherche alors quels sont les opérateurs dans ce langage qui peuvent correspondre à des opérations de généralisation ou de spécialisation utiles pour l'induction, et on construit des algorithmes les utilisant, permettant de résumer les données tout en évitant de les surgénéraliser et d'en tirer des conséquences illégitimes.
- *Les mathématiques appliquées.* Finalement, l'ingénieur peut être tenté de voir dans l'apprentissage un cas particulier de *Résolution de problème inverse*. Dans le cas d'un problème direct, on se donne une « structure » et on en cherche les conséquences. Par exemple, tel avion est capable de supporter telle charge dans telles conditions. Dans le cas d'un problème inverse, on se donne des spécifications sur les capacités souhaitées et on cherche à concevoir un objet qui les vérifie. C'est évidemment typiquement le problème auquel sont confrontés les ingénieurs. Prenons deux exemples :
 - On peut dire que la théorie des probabilités est une théorie s'attachant à un problème direct (étant donné un modèle paramétré, quelles sont les probabilités associées à tel événement ?), tandis que la théorie des statistiques s'attaque à un problème inverse (étant donné un échantillon de données, quel modèle permet de l'expliquer, c'est-à-dire peut l'avoir produit ?).
 - Étant donnés deux nombres, il est facile d'en trouver le produit (problème direct), il est en revanche généralement impossible de trouver à partir d'un nombre ceux dont il est le produit (problème inverse).

Les problèmes inverses sont ainsi souvent des problèmes que l'on dits *mal posés*, c'est-à-dire n'ayant pas de solution unique. Selon cette perspective, l'étude de l'apprentissage peut être vue comme celle des conditions permettant de résoudre un problème mal posé, c'est-à-dire des contraintes qu'il faudra ajouter pour que la procédure de résolution puisse trouver une solution particulière.

Par rapport aux structures de données ou types d'hypothèses visées

Il arrive fréquemment que l'on impose le type de structure de données (ou de *langage d'expression d'hypothèses*) qui doit être recherché par le système apprenant. Cela permet de guider à la fois la détermination de l'algorithme d'apprentissage à utiliser, mais aussi les données qui seront nécessaires pour que l'apprentissage soit possible. Sans chercher à être exhaustif, nous y reviendrons plus longuement au chapitre 3, nous citerons parmi les principales structures de données étudiées :

- Les *expressions booléennes*, qui sont souvent appropriés pour apprendre des concepts définis sur un *langage attribut-valeurs* (par exemple des règles de système expert).
- Les *grammaires et les processus markoviens*, permettant de représenter des séquences d'événements.
- Les *fonctions linéaires ou non linéaires* permettant de discriminer des formes appartenant à un sous-espace ou à son complémentaire.
- Les *arbres de décision* qui permettent l'expression de classifications par des hiérarchies de questions. L'arbre de décisions correspondant est souvent à la fois concis et compréhensible.
- Les *programmes logiques* auxquels il faut songer lorsque l'on cherche à apprendre des concepts relationnels.
- Les *réseaux bayésiens* permettant à la fois de représenter des univers structurés par des relations de causalité et de prendre en compte et d'exprimer des mesures de certitude ou de confiance.

Parfois l'apprentissage peut consister à *changer de structure de données* pour en trouver une équivalente mais plus efficace du point de vue computationnel. C'est encore une fois, sous un autre angle, le problème de l'optimisation de performance.

1.4.3 L'induction vue comme une estimation de fonction

Après avoir brièvement passé en revue les facteurs et les points de vue en jeu dans l'apprentissage artificiel, nous allons maintenant esquisser la manière dont est envisagé actuellement le processus d'apprentissage. Nous allons considérer la tâche de l'apprenant, ainsi que l'approche suivie pour la mener à bien. Il s'agit ici d'un premier exposé qui sera précisé, de manière plus formelle dans le chapitre 2 et dans son complément, le chapitre 17, et qui permettra de comprendre le fonctionnement des algorithmes d'apprentissage décrits dans les chapitres suivants.

Revenons sur le scénario esquissé dans la section 1.4.1 et dans la figure 1.5. Nous supposons que l'environnement, qu'il soit mesuré par les senseurs d'un robot ou qu'il s'exprime sous la forme d'une base de données, fournit un ensemble de formes \mathbf{x}_i définies sur l'espace des entrées \mathcal{X} et tirées aléatoirement suivant une distribution de probabilités notée $\mathcal{D}_{\mathcal{X}}$ (on parle de tirage indépendant et identiquement distribué ou *tirage i.i.d.*). On peut ainsi imaginer qu'une webcam prenne des images à intervalles réguliers d'un carrefour à New-York, et que les formes \mathbf{x}_i mesurées correspondent aux véhicules observés. On pourra supposer que ces véhicules sont indépendants les uns des autres (sauf dans le cas de cortèges officiels ou mortuaires), mais que leur distribution dépendra de la ville, New-York se révélant sans doute différent de Londres ou Nouakchott⁷ sous cet aspect.

Dans le cadre de l'apprentissage supervisé, nous supposons également qu'un oracle étiquette les formes \mathbf{x}_i grâce à une fonction inconnue de l'apprenant, que nous appellerons *fonction cible*,

7. Capitale de la République islamique de Mauritanie.

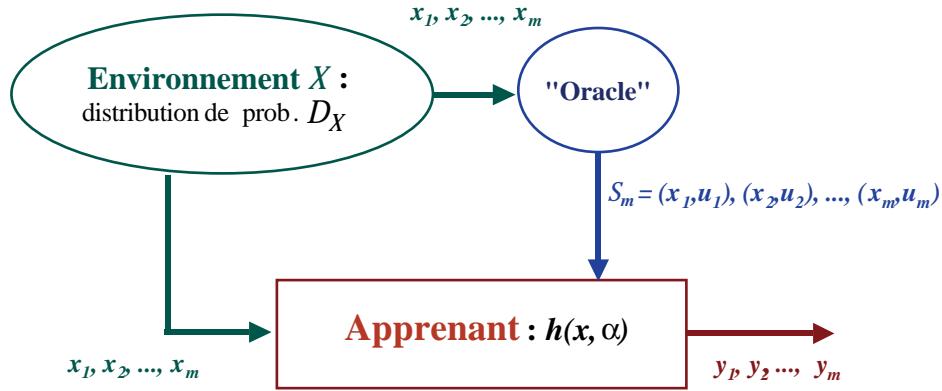


FIG. 1.5 – Le scénario classique de l'apprentissage par induction. L'environnement fournit des données \mathbf{x}_i tirées aléatoirement suivant une distribution $D_{\mathcal{X}}$ sur l'espace d'entrée \mathcal{X} . Ces données sont étiquetées par un oracle qui utilise pour ce faire une fonction $f \in \mathcal{F}$. L'apprenant reçoit un échantillon d'exemples ou couples $(\mathbf{x}_i, \mathbf{u}_i) = (\mathbf{x}_i, f(\mathbf{x}_i))$, et à partir de cet échantillon, doit chercher à deviner f , ou au moins à en trouver une approximation h .

notée f . L'apprenant reçoit donc un ensemble d'exemples

$$\mathcal{S} = \{\mathbf{x}_i, \mathbf{u}_i\}_{1 \leq i \leq m} = \{\mathbf{x}_i, f(\mathbf{x}_i)\}_{1 \leq i \leq m}$$

(voir figure 1.5). On supposera que l'oracle choisit la fonction cible f dans une famille de fonctions notée \mathcal{F} . On supposera également ici, qu'une fois sélectionnée par l'oracle, la fonction f reste constante, de même que la distribution $D_{\mathcal{X}}$ des formes \mathbf{x}_i . Nous étudierons plus tard le cas d'une fonction cible bruitée ou présentant une dérive au cours du temps, ainsi que le cas d'une distribution variable des exemples.

Cette description de l'apprentissage, plus précisément de l'induction, conduit naturellement à voir l'apprentissage comme une tâche d'estimation de fonction à partir d'un échantillon de son comportement. Il s'agit là effectivement du point de vue dominant actuellement.

Notons que le cadre de l'estimation de fonction cible est très général puisqu'il couvre de nombreuses tâches d'apprentissage classiques. Par exemple :

- Un *problème de régression* peut être vu comme un problème d'estimation dans lequel il s'agit de trouver une fonction h telle que :

$$\forall \mathbf{x} \in \mathcal{X}, h(\mathbf{x}) \approx f(\mathbf{x}) = \mathbf{u}$$

- L'*apprentissage d'une classification* d'objets (par exemple apprendre à classer des images de pièces de mobilier en types de meubles, des volatiles caractérisés par un certain nombre d'attributs en types d'oiseaux, etc.) peut-être vu comme l'estimation d'une fonction à valeur discrète, où à chaque entrée correspond une valeur correspondant à une classe. L'apprentissage de la reconnaissance des lettres de l'alphabet peut ainsi être abordé comme l'estimation d'une fonction définie sur un espace d'entrée (par exemple une matrice de pixels) vers un espace de sortie à 26 valeurs.
- L'*apprentissage de concept*, dans lequel il s'agit d'apprendre à reconnaître une classe d'objets parmi tous les autres objets, peut être considéré comme l'estimation d'une fonction binaire (on dit aussi fonction indicatrice), prenant la valeur 1 quand la forme d'entrée est de la classe cible et 0 autrement.

- Un *problème d'optimisation multicritères* dans lequel on cherche à optimiser à la fois plusieurs critères objectifs peut être vu comme l'estimation d'une fonction multi-valuée.

Toujours pour simplifier, nous supposerons que l'apprenant cherche une approximation de la fonction cible à l'intérieur d'une famille \mathcal{H} de *fonctions hypothèses* h . C'est le cas par exemple d'un apprenant utilisant un réseau de neurones dont l'architecture contraint le type de fonctions réalisables à un certain ensemble de fonctions. Par exemple aussi, le désormais fameux avimateur novice ne cherche que des séparations linéaires dans l'espace des descriptions des oies et des cygnes.

Pour définir le problème d'apprentissage, il faut maintenant se donner un critère de performance. On évaluera la qualité de l'estimation h relativement à une espérance de performance dans un environnement donné. C'est pourquoi on spécifie généralement *le critère de performance* d'une fonction hypothèse h sous la forme d'une expression exprimant ce que coûtera le choix de la fonction hypothèse h si la vraie fonction inconnue est f . Par exemple, la performance d'un système d'apprentissage de diagnostic sera mesurée par l'espérance de coût de la décision $h(\mathbf{x})$ lorsque la vraie pathologie est $f(\mathbf{x})$. C'est ce que dénote l'équation suivante :

$$R(h) = \int_{\mathbf{x} \in X, \mathcal{D}_{\mathcal{X}}} l(h(\mathbf{x}), f(\mathbf{x})) d\mathbf{x} \quad (1.1)$$

dans laquelle $R(h)$ dénote une *fonction de risque*, tandis que l désigne une *fonction de perte* définie pour chaque exemple. L'intégrale est prise sur l'ensemble des formes $\mathbf{x} \in \mathcal{X}$ possibles suivant la distribution donnée $\mathcal{D}_{\mathcal{X}}$.

Par exemple, si l'apprenant se trouve à New-York, la distribution des voitures de couleur jaune est différente de celle trouvée à Londres. En supposant que le problème soit d'apprendre à reconnaître des taxis, il faut prendre en compte la distribution des véhicules dans l'environnement d'apprentissage. On suppose naturellement que cette distribution des formes est aussi celle qui sera rencontrée après l'apprentissage. C'est pourquoi cette distribution apparaît dans l'expression du risque. (Apprendre à reconnaître des taxis dans New-York peut se révéler d'une utilité limitée si l'on doit ensuite se débrouiller à Londres, ou plus encore en Mauritanie). Ce serait un nonsens de fournir un échantillon de données non représentatif de l'environnement qui sera rencontré ensuite par l'apprenant et de lui demander d'en tirer une information qui le rende performant dans ce nouvel environnement inconnu pour lui.

La fonction de risque (1.1) mesure donc l'espérance de perte *dans un environnement donné*, spécifié par la distribution $\mathcal{D}_{\mathcal{X}}$ des événements mesurables par l'apprenant.

Pour sélectionner une fonction hypothèse h , l'apprenant doit se fonder sur l'information apportée par chaque exemple $\{\mathbf{x}_i, \mathbf{u}_i\}$ qu'il peut comparer à la prédiction de la fonction d'hypothèse $h(\mathbf{x}_i)$.

Nous avons défini la tâche d'apprentissage comme celui d'un problème d'estimation de fonction à partir de l'observation d'un échantillon de données. Nous nous tournons maintenant vers les principes permettant de réaliser cette estimation.

1.5 L'induction comme un jeu entre espaces

Dans le but de simplifier toute la discussion qui suit et de permettre une visualisation aisée des problèmes, nous nous focalisons dans cette section sur l'apprentissage supervisé de *concept*, c'est-à-dire sur l'apprentissage de *fonctions indicatrices* ou encore à valeur dans $\{0, 1\}$. Les notions

abordées seront cependant d'une portée beaucoup plus générale et valables pour l'essentiel dans toutes les situations d'apprentissage.

L'*apprentissage supervisé de concept* consiste à chercher une fonction $f : \mathcal{X} \rightarrow \{0, 1\}$, c'est-à-dire un étiquetage de chaque forme $\mathbf{x} \in \mathcal{X}$ par 0 (« \mathbf{x} n'appartient pas au concept visé ») ou 1 (\mathbf{x} « appartient au concept »)⁸. Cette fonction est apprise à partir d'un échantillon de points étiquetés que l'on appelle *échantillon d'apprentissage*. Nous noterons

$$\mathcal{S} = \{(\mathbf{x}_1, \mathbf{u}_1), (\mathbf{x}_2, \mathbf{u}_2), \dots, (\mathbf{x}_m, \mathbf{u}_m)\}$$

un échantillon d'apprentissage de m points non nécessairement tous distincts (lorsqu'il sera important de préciser la taille de l'échantillon d'apprentissage, nous le noterons \mathcal{S}_m). Pour des raisons évidentes, on appelle souvent *exemples* ou *exemples positifs* les points étiquetés par 1 ou par '+', et *contre-exemples* ou *exemples négatifs* les points étiquetés par 0 ou par '-'. Il arrivera cependant dans la suite de l'ouvrage que nous parlions d'exemples pour dénoter les points étiquetés, qu'ils le soient positivement (exemples au sens propre) ou négativement (contre-exemples). La figure 1.6 schématisse la tâche d'apprentissage de concepts.

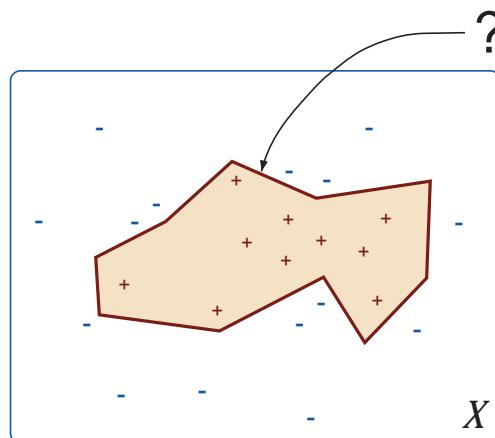


FIG. 1.6 – À partir d'un échantillon de points étiquetés, ici figurés par des points '+' et des points '-', l'apprenant cherche une partition de \mathcal{X} permettant de discriminer les formes \mathbf{x} appartenant au concept de celles n'y appartenant pas.

Nous supposons maintenant, d'une part que l'échantillon d'apprentissage n'est pas bruité, c'est-à-dire que les exemples sont correctement décrits et correctement étiquetés, d'autre part qu'il est n'est pas incohérent, au sens où la même forme n'est pas à la fois exemple et contre-exemple.

Dans ce cadre, l'échantillon d'apprentissage $\mathcal{S} = \{(\mathbf{x}_1, \mathbf{u}_1), (\mathbf{x}_2, \mathbf{u}_2), \dots, (\mathbf{x}_m, \mathbf{u}_m)\}$ fournit une information *cohérente* ou encore *consistante* (un anglicisme qui s'est introduit dans le jargon de l'apprentissage artificiel mais que, pour notre part, nous éviterons) à l'apprenant dans la mesure où la partie de \mathcal{X} qu'il cherche doit couvrir tous les exemples positifs de l'échantillon (ce que l'on appelle la propriété de *complétude*) et ne couvrir aucun des exemples négatifs (ce que l'on appelle la propriété de *correction*).

Dans ce cadre restreint, on peut maintenant poser deux questions :

- Quelle information est fournie par chaque exemple ?

8. Ces deux classes sont aussi notées $\{+, -\}$.

- Comment, sur la base de l'échantillon d'apprentissage, choisir une hypothèse, c'est-à-dire dans le cas de l'estimation d'une fonction indicatrice, une partition de \mathcal{X} ?

1.5.1 L'apprentissage est impossible...

Dans le cadre de l'induction de concept, donc d'une fonction indicatrice définie sur l'espace \mathcal{X} des entrées, l'apprentissage revient à chercher une partition de l'espace \mathcal{X} . En effet, il s'agit d'identifier les régions de \mathcal{X} , donc les formes \mathbf{x} , correspondant au concept visé (voir figure 1.6).

Que peut nous apprendre un échantillon d'exemples \mathcal{S} sur cette partition ?

Supposons que l'apprenant soit prêt à considérer toutes les partitions possibles de \mathcal{X} , donc que n'importe quel étiquetage des formes $\mathbf{x} \in \mathcal{X}$ soit possible *a priori*. Cela signifie que si le cardinal de \mathcal{X} , $|\mathcal{X}|$, est fini, il existe $2^{|\mathcal{X}|}$ partitions possibles de \mathcal{X} .

Supposons alors que nous cherchions à déterminer la classe d'un point $\mathbf{x} \in \mathcal{X}$ inconnu connaissant la classe de tous les points d'apprentissage $\mathbf{x}_i \in \mathcal{X}$. Comment procéder ?

Puisque nous manipulons des partitions de \mathcal{X} , nous pourrions considérer toutes les partitions cohérentes avec l'échantillon d'apprentissage, puis décider alors de la classe de \mathbf{x} en fonction de ces partitions. Si toutes les partitions cohérentes avec l'échantillon \mathcal{S} prescrivent que \mathbf{x} appartient au concept, ou au contraire n'y appartient pas, cela déterminera notre décision pour la classe de \mathbf{x} . Supposons même que toutes ces partitions ne soient pas d'accord sur la classe de \mathbf{x} , nous pourrions encore décider que la classe de \mathbf{x} est la classe majoritaire parmi les prédictions de toutes les partitions cohérentes avec l'échantillon d'apprentissage.

Malheureusement, aucun de ces deux cas de figure ne se présente. Il se trouve que si l'on prend toutes les partitions cohérentes avec n'importe quel ensemble de points d'apprentissage \mathcal{S} (c'est-à-dire prédisant correctement l'étiquette de chacun de ces points), et si l'on prend n'importe quel point $\mathbf{x} \notin \mathcal{S}$, alors il existe autant de partitions prédisant l'étiquette 1 pour \mathbf{x} que de partitions prédisant l'étiquette 0. L'échantillon d'apprentissage à lui tout seul ne fournit donc pas une base suffisante pour décider de la classe d'un point nouveau. L'induction, c'est-à-dire l'extrapolation du connu à l'inconnu est impossible. Seul un apprentissage par cœur est réalisable.

Les deux questions soulignées dans la section précédente ont donc reçu une réponse qui jette pour le moins une ombre sur la possibilité de l'induction. Chaque exemple ne fournit aucune information sur une forme inconnue. Toutes les partitions de l'espace \mathcal{X} cohérentes avec l'échantillon sont également probables et leurs prédictions s'annulent en chaque point inconnu. L'aventure de l'apprentissage artificiel tournerait-elle court ?

Exemple 1 (Apprentissage de fonction booléenne (1))

Soit un ensemble \mathcal{X} de points décrits par n attributs binaires. Chaque partition de \mathcal{X} correspond à un étiquetage particulier des 2^n points de \mathcal{X} . Il existe donc 2^{2^n} partitions différentes de \mathcal{X} ou encore 2^{2^n} fonctions indicatrices définies de \mathcal{X} sur $\{0,1\}$.

Supposons que l'échantillon d'apprentissage comporte m exemples distincts. Le nombre de partitions de \mathcal{X} compatibles avec ces m exemples est : 2^{2^n-m} puisque m points sur les 2^n sont fixés.

Prenons le cas de $n = 10$ attributs binaires et de $m = 512$ exemples d'apprentissage. Le cardinal de \mathcal{X} est $|\mathcal{X}| = 2^{10}$, soit 1024 points différents, ce qui n'est pas un espace très grand. Il existe 2^{1024} manières différentes de les étiqueter par 1 ou 0. Après l'observation de la moitié de ces 1024 points, il reste $2^{1024-512}$ partitions possibles, soit 2^{512} . On voit que ces 512 exemples laissent un ensemble considérable de partitions possibles.

Étudions un problème plus simple dans lequel les exemples sont décrits par trois attributs binaires. Cela fait $2^3 = 8$ formes possibles. Supposons que cinq exemples parmi ces huit aient

x_1	x_2	x_3	$f(\mathbf{x})$
0	0	0	+
0	0	1	-
0	1	0	+
0	1	1	?
1	0	0	+
1	0	1	?
1	1	0	?
1	1	1	-

FIG. 1.7 – Soit f une fonction binaire définie sur un espace d'entrée à trois attributs. La table fournit un échantillon de 5 exemples de cette fonction.

été étiquetés par l'oracle, comme le montre la table 1.7. Pour fixer complètement une fonction, il faut déterminer la valeur des trois dernières formes. Il faut donc faire un choix entre $2^3 = 8$ fonctions. Supposons que nous voulions déterminer la valeur associée à l'entrée $(0 \ 1 \ 1)$. Il y a quatre fonctions parmi les huit qui sont associées à la sortie + et quatre associées à la sortie -. Il est donc impossible d'avoir même seulement une préférence pour une prédiction plutôt qu'une autre concernant l'étiquette de ce point.

Nous nous sommes placés dans le cas où l'apprenant cherche directement une partition de l'espace d'entrée \mathcal{X} , c'est-à-dire qu'il cherche à déterminer l'étiquette de chaque forme $\mathbf{x} \in \mathcal{X}$. C'est évidemment impossible, sauf dans le cas d'espaces \mathcal{X} très restreints pour lesquels un apprentissage par cœur est envisageable. En d'autres termes, il est généralement impossible d'apprendre une partition de \mathcal{X} en extension, c'est-à-dire en énumérant toutes les formes et leur étiquette associée.

1.5.2 ... sans limiter l'espace des hypothèses

C'est pourquoi on utilise généralement pour décrire des partitions de \mathcal{X} un *langage de description des hypothèses*, que nous noterons $\mathcal{L}_{\mathcal{H}}$. Celui-ci permet de définir un espace d'expressions ou d'hypothèses \mathcal{H} , par exemple l'espace des hypothèses décrites par une conjonction de conditions sur les descripteurs⁹.

Ainsi, dans l'exemple précédent, on pourrait décrire des fonctions binaires du type $(x_1 = 0) \wedge (x_2 = 1) \wedge (x_3 = 1)$. En revanche, ce langage interdit de considérer une fonction telle que $(x_1 = 0 \wedge x_2 = 1 \wedge x_3 = 1) \vee (x_1 = 0 \vee x_2 = 0 \wedge x_3 = 0)$.

La figure 1.8 donne des exemples de la restriction de l'espace des hypothèses par un langage de description.

Lorsqu'un espace d'hypothèses \mathcal{H} est disponible, la recherche d'une partition de \mathcal{X} s'effectue par l'intermédiaire de \mathcal{H} . Il s'agit de chercher dans \mathcal{H} , une hypothèse h correspondant à une partition de \mathcal{X} appropriée.

Les avantages de l'utilisation explicite d'un espace d'hypothèses sont multiples :

1. D'abord, grâce au langage $\mathcal{L}_{\mathcal{H}}$, l'apprenant manipule des partitions de \mathcal{X} en intension et non plus en extension. Il travaille sur des expressions du langage $\mathcal{L}_{\mathcal{H}}$ et non pas sur des ensembles définis par l'énumération de leurs éléments.
-
9. Nous verrons au chapitre (3) qu'il s'agit du langage CNF (Conjunctive Normal Form).

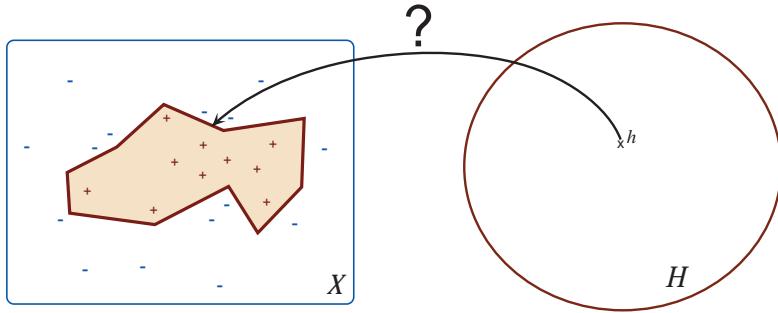


FIG. 1.8 – *Introduction d'un espace d'hypothèses \mathcal{H} . Chaque point de \mathcal{H} , ou encore hypothèse, correspond à une partition de l'espace des entrées \mathcal{X} .*

2. Ensuite, et c'est un point capital d'après la discussion de la section précédente, il devient possible d'effectuer une *induction* à partir d'un échantillon limité d'exemples. Il suffit pour cela que $\mathcal{L}_{\mathcal{H}}$ ne permette pas de décrire toutes les partitions de \mathcal{X} .

Voyons pourquoi.

Nous allons d'abord le montrer en reprenant l'exemple précédent.

Exemple 2 (Apprentissage de fonction booléenne (2))

Supposons que pour une raison quelconque, l'apprenant qui reçoit des entrées décrites sur les trois descripteurs binaires x_1, x_2, x_3 ne puisse prendre en compte en fait que le premier et le troisième descripteurs, c'est-à-dire x_1 et x_3 , pour décider de l'étiquette de la forme reçue. Cela revient à dire que le nombre de fonctions que l'apprenant peut considérer est de 4 (2^2) au lieu des 8 (2^3) possibles lorsque l'on prend en compte les trois descripteurs. Cela signifie en particulier que si l'échantillon d'apprentissage contient les exemples $(000) \rightarrow -$ et $(010) \rightarrow +$, l'apprenant ne pourra pas construire une hypothèse, c'est-à-dire une fonction, qui permette d'en rendre compte.

En revanche, cette fois-ci, l'échantillon d'apprentissage fourni dans la table précédente lui permet de faire une prédiction pour le point $(0\ 1\ 1)$. Ceci parce que la seule fonction à valeur sur x_1, x_3 et cohérente avec les exemples d'apprentissage est la fonction dont le tableau est le suivant :

x_1	x_3	$f(\mathbf{x})$
0	0	+
0	1	-
1	0	+
1	1	-

Et selon cette fonction, l'étiquette de la forme $(0\ 1\ 1)$ est '-'.

Nous voyons donc qu'une limitation de l'espace d'hypothèses rend possible l'induction. Naturellement, ce pouvoir a un prix. Si les « œillères » dont on a muni l'apprenant ne correspondent pas avec la fonction cible de la Nature ou de l'oracle, on ne peut pas l'apprendre correctement.

La figure 1.9 est également une illustration de cette même idée.

Pour qualifier ces « œillères » qui limitent l'espace des fonctions hypothèses que peut considérer l'apprenant, on parle d'un *biais de représentation*. Évidemment, tout biais de

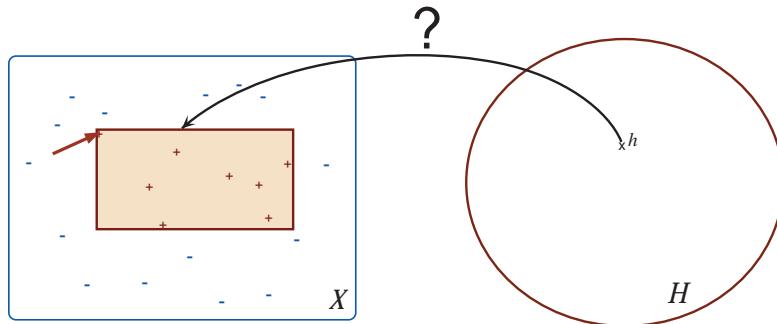


FIG. 1.9 – Supposons que le langage de représentation des hypothèses \mathcal{L}_H corresponde à une restriction aux parties de \mathcal{X} qui sont des rectangles. Dans ce cas, la donnée du point '+' indiqué par la flèche implique que tous les points inscrits dans le rectangle dont il délimite un angle sont de classe '+'. On voit que dès lors, il devient possible d'induire la classe de points jamais observés dans l'échantillon d'apprentissage. Par exemple, selon ce biais, le point dénoté par un rond noir est prédict appartenir à la classe '+'.

représentation correspond à un « acte de foi » sur le type d'hypothèses adéquat pour décrire le monde. Cet acte de foi peut être erroné auquel cas l'apprentissage peut donner de très mauvais résultats (voir figure 1.10). Il faudra arriver à détecter quand c'est le cas.

Nous verrons plus loin que la notion de *biais* en apprentissage se définit comme toute restriction de l'ensemble des hypothèses potentielles, y compris des restrictions qui vont plus loin que les restrictions portant sur le langage d'expression des hypothèses.

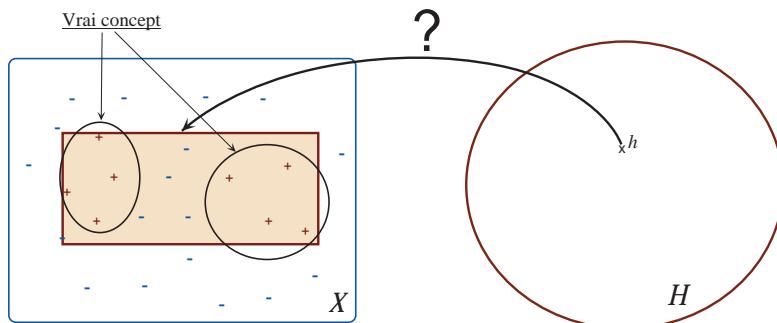


FIG. 1.10 – Supposons que le langage de représentation des hypothèses \mathcal{L}_H corresponde à une restriction aux parties de \mathcal{X} qui sont des rectangles et que la partition « vraie » de la Nature, correspondant aux exemples positifs, soit représentée par les deux « patatoïdes ». Dans ce cas, il est impossible d'approximer correctement le concept cible à l'aide d'une hypothèse de \mathcal{H} .

3. Finalement, l'espace \mathcal{H} des hypothèses peut offrir des structures permettant son exploration de manière plus ou moins systématique et plus ou moins efficace. En particulier, une relation d'ordre sur \mathcal{H} corrélée avec la généralité de l'induction effectuée est très utile (voir le chapitre 4).

1.5.3 L'exploration de l'espace des hypothèses

Soit un espace d'hypothèses \mathcal{H} , un espace d'entrée \mathcal{X} et un échantillon d'apprentissage $\mathcal{S} = ((\mathbf{x}_1, \mathbf{u}_1), (\mathbf{x}_2, \mathbf{u}_2), \dots, (\mathbf{x}_m, \mathbf{u}_m))$. La tâche de l'apprenant est de trouver une hypothèse h approximant au mieux, au sens d'une certaine mesure de performance, une fonction cible f sur la base de l'échantillon \mathcal{S} dans lequel on suppose que chaque étiquette \mathbf{u}_i a été calculée grâce à la fonction f appliquée à la forme \mathbf{x}_i .

Comment trouver une telle hypothèse $h \in \mathcal{H}$? Deux questions se posent :

1. Comment savoir qu'une hypothèse satisfaisante (voire optimale) a été trouvée, et plus généralement comment évaluer la qualité d'une hypothèse?
2. Comment organiser la recherche dans \mathcal{H} ?

Quel que soit le processus guidant l'exploration de \mathcal{H} , il est nécessaire que l'apprenant puisse évaluer les hypothèses h qu'il considère à un instant t de sa recherche. Nous avons vu (section 1.4.2.1) que cette évaluation fait intervenir une fonction de coût interne (par exemple un écart quadratique entre les sorties calculées à partir de h et les sorties désirées \mathbf{u} fournies dans l'échantillon d'apprentissage). C'est cette fonction de coût, plus, éventuellement, d'autres informations fournies par l'environnement (y compris l'utilisateur par exemple), qui permet à l'apprenant de mesurer sa performance sur l'échantillon d'apprentissage et de décider s'il doit poursuivre sa recherche dans \mathcal{H} où s'il peut s'arrêter.

Par exemple, dans le cas de l'apprentissage supervisé de concept, en supposant des descriptions non bruitées des entrées, l'apprenant cherche une hypothèse exprimable dans le langage $\mathcal{L}_{\mathcal{H}}$ couvrant tous les exemples positifs de l'échantillon d'apprentissage et ne couvrant aucun des exemples négatifs.

La figure 1.11 schématise la recherche d'une hypothèse dans le cas d'un apprentissage hors ligne (quand tout l'échantillon d'apprentissage est supposé d'emblée disponible). La figure 1.12 est relative à un apprentissage en ligne, dans lequel les exemples sont fournis séquentiellement. Dans ce dernier cas, on suppose ici que l'hypothèse courante h_t est comparée à l'entrée courante $\mathbf{z}_{t+1} = (\mathbf{x}_{t+1}, \mathbf{u}_{t+1})$ et modifiée s'il y a lieu.

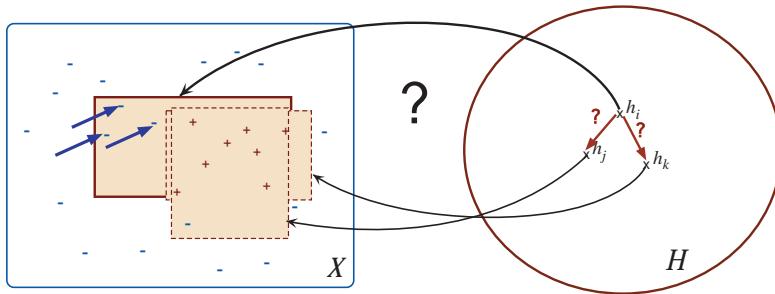


FIG. 1.11 – Si l'hypothèse courante h_t est insatisfaisante –ici elle n'exclue pas tous les exemples négatifs connus–, alors il faut que l'apprenant cherche une nouvelle hypothèse dans \mathcal{H} . La question est : où doit-il chercher ?

En supposant qu'à l'instant t , l'apprenant juge insatisfaisante son hypothèse courante h_t , comment peut-il en changer? C'est là que se décide l'efficacité de l'apprentissage et que joue la structure exploitable sur l'espace \mathcal{H} . Plus celle-ci sera riche et fine, et plus il sera envisageable

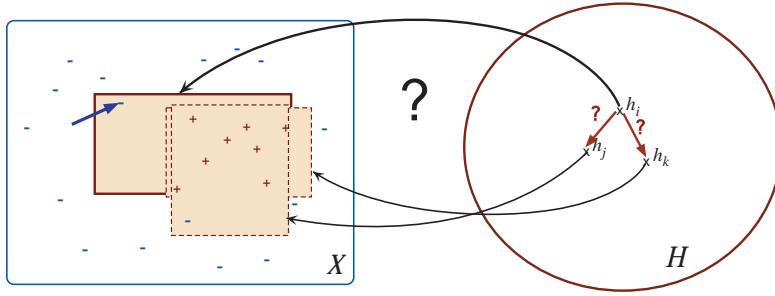


FIG. 1.12 – Si l'hypothèse courante h_t est insatisfaisante (ici elle ne couvre pas le nouvel exemple $z_{t+1} = (\mathbf{x}_{t+1}, \mathbf{u}_{t+1})$), alors il faut que l'apprenant cherche une nouvelle hypothèse dans \mathcal{H} . Encore une fois, la question est : où doit-il chercher ?

d'organiser efficacement l'exploration de \mathcal{H} . Examinons rapidement trois possibilités par ordre croissant de structuration.

- L'espace \mathcal{H} des hypothèses ne présente *aucune structure*. Dans ce cas, seule une exploration aléatoire est possible. Rien ne permet de guider la recherche, ni même de tirer parti des informations déjà glanées sur \mathcal{H} . C'est le cas où l'on ne connaît rien *a priori* sur \mathcal{H} .
- *Une notion de voisinage est définissable sur \mathcal{H}* . Il est alors possible d'opérer une exploration par des techniques d'optimisation comme le gradient¹⁰. L'avantage de ces techniques, et ce qui les rend si populaires, c'est qu'elles sont d'un usage très général puisqu'il est souvent possible de définir une notion de voisinage sur un espace. Un problème fondamental est celui de la pertinence de cette notion de voisinage. Une mauvaise relation de voisinage peut en effet éloigner l'apprenant des régions prometteuses de l'espace ! Par ailleurs, c'est encore une structure faible qui, sauf dans des cas particuliers (différentiabilité, convexité, etc. de la fonction à optimiser) ne permet pas une exploration rapide.
- Il est parfois possible de disposer d'une structure plus forte permettant d'organiser l'exploration de \mathcal{H} . C'est le cas en particulier des *structures d'ordre partiel induites par des relations de généralité entre hypothèses*. Dans ce cas, par exemple, il devient possible de modifier une hypothèse erronée en la spécialisant juste assez pour qu'elle ne couvre plus le nouvel exemple négatif, ou au contraire en la généralisant juste assez pour qu'elle couvre le nouvel exemple positif fourni. Ce type d'exploration, possible en particulier quand l'espace des hypothèses est structuré par un langage (voir chapitre 4), est généralement mieux guidé et plus efficace qu'une exploration aveugle.

De ce qui précède, il est évident que plus la structuration de l'espace des hypothèses est forte et adaptée au problème d'apprentissage, et plus les connaissances *a priori*, s'exprimant en particulier dans les biais et dans le critère de performance, sont importantes, plus l'apprentissage sera facilité. En contrepartie, bien sûr, cela nécessitera un travail de réflexion préalable d'autant plus important.

1.6 Retour sur l'organisation de l'ouvrage

Toute étude de l'apprentissage artificiel peut se situer par rapport à trois pôles :

- Une approche théorique de l'apprentissage s'attachant à identifier ce qu'il est possible d'ap-

10. Ce terme inclut ici des méthodes plus ou moins sophistiquées y compris les approches de type évolution simulée (algorithmes génétiques) et celles des réseaux connexionnistes.

prendre ou, plus précisément, ce qui est nécessaire pour qu'un apprentissage soit possible en principe.

- Une approche d'ingénieur concerné par la réalisation de méthodes d'apprentissage sous formes d'algorithmes et de programmes informatiques.
- Une approche d'utilisateur intéressé par les réalisations des programmes d'apprentissage et les problèmes qu'ils permettent de résoudre.

Nous avons essayé de rendre compte de ces trois points de vue tout au long de l'ouvrage, même si la progression logique impose de partir de prémisses plutôt conceptuelles et théoriques pour aller vers la conception de systèmes d'apprentissage et, de là, vers les applications.

Tout ouvrage général sur l'apprentissage artificiel doit affronter la difficulté d'avoir à présenter une collection de méthodes et d'algorithmes parfois issus de communautés scientifiques différentes, pour des motivations diverses (métaphores biologiques, modèles de la physique, architecture cognitive,...) et souvent décrits dans les articles scientifiques à l'aide de notations non homogènes. Chaque auteur doit alors faire un choix pour organiser, le moins arbitrairement possible, l'exposition de toutes ces techniques. Parmi nos illustres prédécesseurs, Tom Mitchell [Mit97] a choisi d'équilibrer tout au long de l'ouvrage théorie et pratique, à l'image de son cours à l'université de Carnegie-Mellon (CMU), sans suivre de principe directeur particulier et en assumant les différences de notations entre les écoles de pensée. Pat Langley [Lan96] a fait le pari audacieux de structurer tout son ouvrage sur les langages de représentation des hypothèses manipulés par les systèmes apprenants, en imposant une notation uniforme et des exemples de tâches d'apprentissage illustrant l'ensemble des méthodes. Comme nous l'avons déjà dit dans l'avant-propos, nous avons choisi de suivre un autre principe structurant.

Nous avons décidé de présenter dans une première partie les concepts et principes fondamentaux qui permettent de comprendre et de justifier la plupart des méthodes d'apprentissage. En particulier nous nous sommes attachés à l'étude des conditions sous lesquelles un apprentissage est possible, ou impossible, et ceci indépendamment d'un algorithme particulier. Ceci nous permet de cerner les conditions nécessaires à un apprentissage, ainsi que de motiver l'utilisation de certains principes inductifs que l'on retrouve à la base de toutes les méthodes d'apprentissage. Le reste de l'ouvrage est dédié aux méthodes et algorithmes d'apprentissage ainsi qu'aux réalisations associées. Afin d'en organiser l'exposition, nous avons choisi de centrer notre attention sur le problème de la recherche d'une ou plusieurs hypothèse(s) dans l'espace d'hypothèses \mathcal{H} . Dans la section précédente, nous avons évoqué l'influence des connaissances préalables sur le processus de recherche et son efficacité. Plus l'espace \mathcal{H} se trouve doté d'une structure forte, et plus son exploration peut être guidée, conduisant en général à une plus grande efficacité. L'ouvrage adopte cette ligne directrice en présentant les méthodes d'apprentissage en fonction de la structuration de \mathcal{H} , partant des espaces les mieux structurés, pour aller graduellement vers l'apprentissage dans des espaces « minimaux » pour lesquels il n'existe même plus d'espace d'hypothèses à proprement parler, mais seulement une notion de voisinage dans l'espace \mathcal{X} des entrées et une mesure de performance. Les trois grandes parties présentant ces méthodes regroupent ainsi d'abord les méthodes d'*apprentissage par exploration* suivant les directions de recherche fournies par l'espace d'hypothèses, ensuite les méthodes d'*apprentissage par optimisation* lorsque \mathcal{H} ne dispose plus que d'une notion de voisinage et d'une mesure de performance, et que sont donc utilisables essentiellement des techniques de gradient, finalement les méthodes d'*apprentissage par interpolation* qui sont les seules utilisables quand on ne connaît plus d'espace d'hypothèses *a priori*.

Plus la connaissance préalable est faible, et plus l'apprentissage requiert de données pour aboutir. On ne peut pas gagner sur tous les tableaux. En contrepartie, les méthodes développées

pour les tâches dans lesquelles on dispose de peu d'informations préalables sont aussi celles qui sont d'usage le plus général, s'adaptant à tous les contextes. C'est pourquoi ces méthodes (par exemple les réseaux connexionnistes ou les algorithmes génétiques) sont les plus populaires, prêtes à être essayées sans grands efforts de réflexion *a priori*. Nous avons voulu souligner que ce calcul est parfois mauvais, et qu'il est souvent rentable de chercher à tirer parti de toutes les connaissances disponibles. Par ailleurs, il nous semble aussi que les tâches d'apprentissage essentiellement numériques qui ont fait florès ces dernières années vont probablement bientôt céder le pas à des tâches d'apprentissage – comme la recherche de documents sur le réseau, leur analyse automatique, etc. – requérant des espaces d'hypothèses beaucoup plus structurés et prenant en compte une énorme quantité de connaissances. C'est pourquoi nous avons réservé une place importante à ces méthodes, malgré leur usage encore modéré dans les applications actuelles.

Notes pour aller plus loin

L'apprentissage artificiel est une discipline jeune, à l'instar de l'intelligence artificielle et de l'informatique, mais elle a déjà une histoire. Nous la brossons ici à grands traits, croyant qu'il est toujours intéressant de connaître le passé d'une discipline, et que cela peut révéler, par les tensions mises à jour, les problèmes profonds, les grandes options, les écarts nécessaires.

Il serait bien sûr intéressant d'examiner l'étude de l'apprentissage artificiel dans une perspective plus large, tant historiquement, en remontant l'histoire de la pensée, que par la prise en compte des parrainages exercés par d'autres disciplines: philosophie, psychologie, biologie, logique, mathématique, etc. Cette étude qui reste à faire dépasse cependant de beaucoup le cadre de notre ouvrage et nous nous limiterons à quelques jalons.

Des principes préliminaires théoriques de l'apprentissage sont posés dès les premiers travaux en statistique dans les années 1920 et 1930, cherchant à déterminer comment inférer un modèle à partir de données, mais surtout comment valider une hypothèse par rapport à un jeu de données. Fisher en particulier étudie les propriétés des modèles linéaires et comment ils peuvent être inférés à partir d'un échantillon de données. À la même période, l'informatique naît avec les travaux de Gödel, Church puis surtout Turing en 1936, et les premières simulations informatiques deviennent possibles après la seconde guerre mondiale. À côté des réflexions théoriques et des débats conceptuels sur la cybernétique et le cognitivisme, dont nous avons parlé dans ce chapitre, des pionniers essaient de programmer des machines pour réaliser des tâches intelligentes, intégrant souvent de l'apprentissage. C'est particulièrement le cas des premières simulations de tortues ou souris cybernétiques que l'on place dans des labyrinthes en espérant les voir apprendre à s'en sortir de plus en plus vite. De son côté, Samuel chez IBM, dans les années 1959-1962, développe un programme pour jouer au jeu de dames américain qui apprend une fonction d'évaluation des positions lui permettant de devenir rapidement un très bon joueur.

Dans les années 1960, l'apprentissage est marqué par deux courants. D'une part, un premier connexionnisme, qui sous la houlette de Rosenblatt père du perceptron, voit se développer des petits réseaux de neurones artificiels testés sur des tâches d'apprentissage supervisé de classes d'objets. D'autre part, des outils conceptuels sur la reconnaissance des formes se développent.

À la fin des années 1960, la publication du livre de Minsky et Papert ([MP69]) qui énonce les limites des perceptrons a pour effet d'arrêter pour une quinzaine d'années presque toutes les recherches dans ce domaine. De manière concomitante, l'accent mis en intelligence artificielle, dans les années 1970, sur les connaissances, leur représentation et l'utilisation de règles d'inférence sophistiquées (période des systèmes experts) encourage les travaux sur l'apprentissage dans des systèmes basés sur des représentations des connaissances structurées mettant en jeu

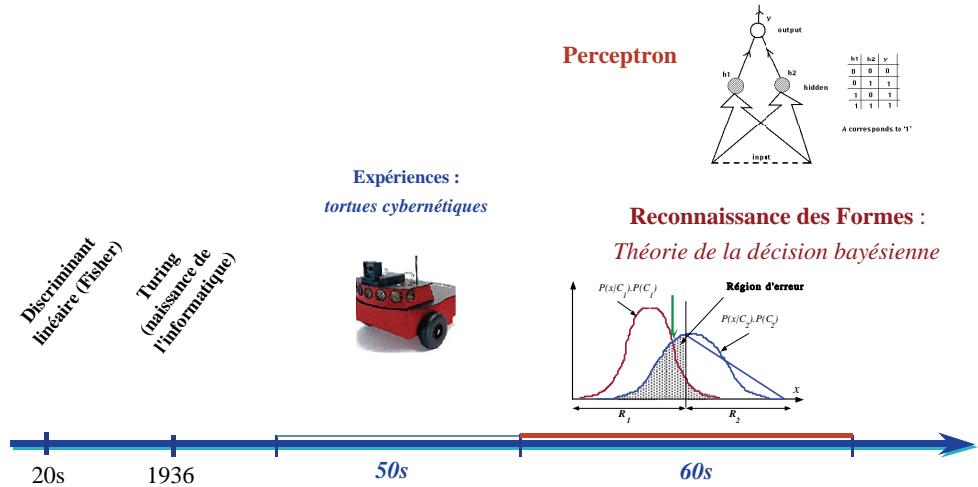


FIG. 1.13 – La première période de l'apprentissage artificiel.

des règles d'inférence complexes comme la généralisation, l'analogie, etc. C'est alors le triomphe de systèmes impressionnantes réalisant des tâches d'apprentissage spécifiques en simulant plus ou moins des stratégies mises en jeu dans l'apprentissage humain. On retiendra en particulier le système ARCH de Winston en 1970 (voir chapitre 2) qui apprend à reconnaître des arches dans un monde de blocs à partir d'exemples et de contre-exemples ; le système AM de Lenat en 1976, qui découvre des conjectures dans le domaine de l'arithmétique par l'utilisation d'un jeu d'heuristiques elles-mêmes apprises dans le système EURISKO du même auteur en 1982, ou bien encore le système META-DENDRAL de Mitchell qui apprend des règles dans un système expert dédié à l'identification de molécules chimiques.

C'est aussi une période durant laquelle le dialogue est facile et fécond entre les psychologues et les praticiens de l'apprentissage artificiel, les hypothèses portant dans les deux communautés sur des concepts comme les mémoires à court terme et à long terme, le type procédural ou déclaratif des connaissances, etc. D'où aussi des systèmes comme ACT de Anderson testant des hypothèses générales sur l'apprentissage de concepts mathématiques dans l'éducation.

Cependant, aussi spectaculaires soient-ils, ces systèmes présentent des faiblesses qui viennent de leur complexité. La première, la moins déterminante mais néanmoins influente, est qu'ils sont à la limite de ce qui est réalisable dans le cadre d'un travail de thèse, c'est-à-dire le *quantum* d'action dans l'institution scientifique. La deuxième est que leur réalisation implique nécessairement un grand nombre de choix, petits et grands, souvent implicites, et qui de ce fait ne permettent pas une réplication aisée des expériences, et surtout jettent le doute sur la portée générale et générique des principes mis en avant. C'est pourquoi les années 1980 ont vu progressivement se tarir les travaux portant sur de telles simulations à quelques brillantes exceptions près comme les systèmes ACT ou SOAR.

De plus, ces années ont vu une réémergence très puissante du connexionnisme en 1985, avec en particulier la découverte d'un nouvel algorithme d'apprentissage par descente de gradient pour les perceptrons multicouche (voir chapitre 10). Cela a profondément modifié l'étude de l'apprentissage artificiel en ouvrant grand la porte à tous les concepts et techniques mathématiques portant sur l'optimisation et sur les propriétés de convergence. Parallèlement à l'intrusion des mathématiques continues, d'autres mathématiciens se sont engouffrés (derrière Valiant en 1984 [Val84a]) dans la brèche ouverte par la notion d'espace des versions due à Mitchell (voir chapitre 4) et qui en gros permet d'envisager l'apprentissage comme la recherche dans un espace

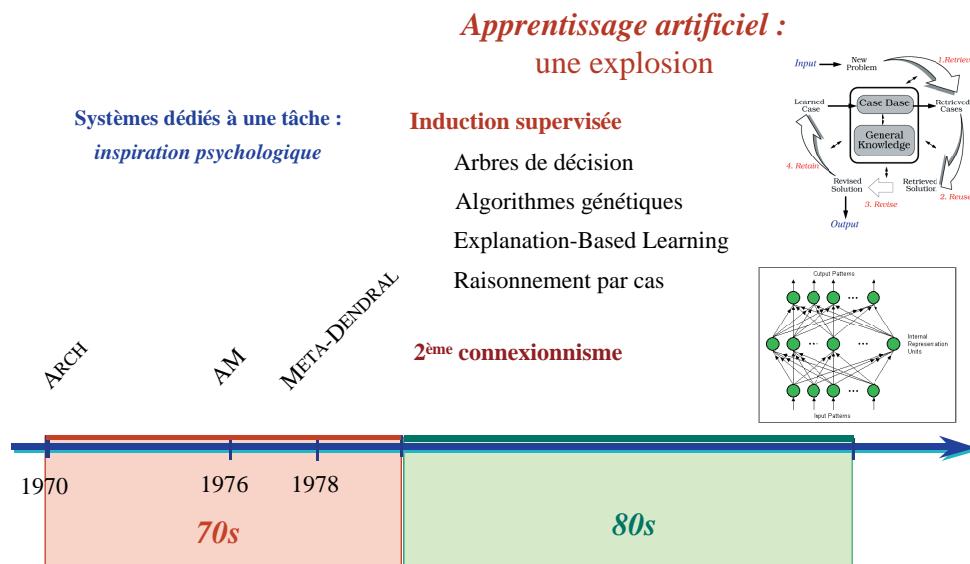


FIG. 1.14 – La deuxième période de l'apprentissage artificiel.

d'hypothèses défini *a priori* d'une hypothèse cohérente avec les données. D'un seul coup l'apprentissage était vu non plus comme la recherche d'algorithmes simulant une tâche d'apprentissage, mais comme un processus d'élimination d'hypothèses ne satisfaisant pas un critère d'optimisation. Il s'agissait alors dans ce cadre de chercher comment un échantillon de données tiré aléatoirement pouvait permettre d'identifier une bonne hypothèse dans un espace d'hypothèses donné. C'était extrêmement déroutant, et comme le langage utilisé dans ces recherches était assez éloigné de celui des praticiens de l'apprentissage artificiel, ceux-ci continuèrent à développer des algorithmes plus simples mais plus généraux que ceux de la décennie précédente : arbres de décision (chapitre 11), algorithmes génétiques (chapitre 8), induction de programmes logiques (chapitre 5), etc.

Ce n'est que dans les années 1990, et surtout après 1995 et la parution d'un petit livre de Vapnik ([Vap95]), que la théorie statistique de l'apprentissage (chapitres 2 et 17) a véritablement influencé l'apprentissage artificiel en donnant un cadre théorique solide à des interrogations et à des constatations empiriques faites dans la pratique de l'apprentissage artificiel.

Le développement actuel de la discipline est dominé à la fois par un effort théorique vigoureux dans les directions ouvertes par Vapnik et les théoriciens de l'approche statistique, et par un redéploiement vers la mise à l'épreuve des techniques développées sur de grandes applications à finalité économique, comme la fouille de données, ou à finalité socio-économiques, comme la génomique. Il est indéniable que pour le moment l'apprentissage est ressenti comme nécessaire dans de très nombreux champs et que nous vivons un âge d'or pour cette discipline. Cela ne doit cependant pas faire oublier les immenses territoires laissés en friche (voir chapitre 17), ni la nécessité de renouer le dialogue avec les psychologues, les didacticiens, et plus généralement tous ceux qui travaillent sur l'apprentissage sous une forme ou une autre.

Les lecteurs intéressés par des articles généraux sur l'apprentissage peuvent se reporter à des articles parus dans des magazines scientifiques, dont : plusieurs numéros hors série de la revue *Science & Vie* : *Le cerveau et l'intelligence* déc. 1991, *À quoi sert le cerveau ?* juin 1996, *Le cerveau et la mémoire* mars 1998, *Les performances de la mémoire humaine* sept. 2000 ; des numéros hors série de la revue *La Recherche* : *L'intelligence artificielle* oct. 1985, *La mémoire* juil. 1994, *L'intelligence* déc. 1998, *La mémoire et l'oubli* juil. 2001 ; un numéro hors série de la

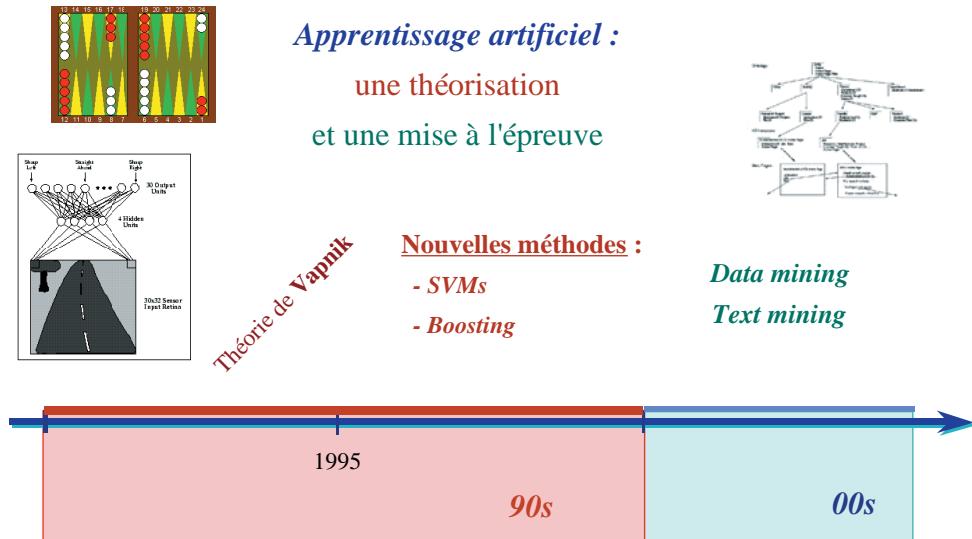


FIG. 1.15 – La troisième période de l'apprentissage artificiel.

revue Pour la Science : *La mémoire : le jardin de la pensée* avril 2001.

Une large littérature existe sur les fondements philosophiques de l'intelligence artificielle ou sur son histoire, dont une part non négligeable provient de travaux français. Nous citons ici des ouvrages faciles à se procurer, mais nous encourageons le lecteur à faire des recherches plus amples (voir [And92, BKL00, Cre97, Dup94, Eng96, Pin97]).

Voici une liste non exhaustive de revues spécialisées sur l'apprentissage artificiel :

- *Machine Learning journal*
- *Journal of Machine Learning Research* (disponible gratuitement sur <http://www.ai.mit.edu/projects/jmlr/>)
- *Journal of Artificial Intelligence Research* (JAIR) accessible gratuitement sur Internet (<http://www.ai.mit.edu/projects/jmlr/>)
- *Data Mining and Knowledge Discovery journal*
- *Transactions on Knowledge and Data Engineering*

Voici aussi une liste de conférences complètement dédiées à l'apprentissage artificiel. Beaucoup d'autres dans le domaine de l'intelligence artificielle, de la reconnaissance des formes et de la fouille de données sont aussi pertinentes :

- *International Conference on Machine Learning* (ICML) : conférence annuelle internationale (mais dominée par les Américains).
- *European Conference on Machine Learning* (ECML) : conférence annuelle européenne (mais internationale).
- *Conférence francophone d'Apprentissage* (CAP) : conférence annuelle francophone qui a pris la suite des Journées Françaises d'Apprentissage (JFA) depuis 1999.

Résumé

Il existe plusieurs types ou problèmes d'apprentissage qui sont définis par un certain nombre de caractéristiques dont l'espace des données, l'espace des hypothèses et le protocole régissant les interactions de l'apprenant avec son environnement. On distingue particulièrement l'apprentissage supervisé pour lequel un oracle fournit les réponses désirées, l'apprentissage non supervisé et l'apprentissage par renforcement. L'étude de l'apprentissage tourne en particulier autour de deux questions :

- L'apprentissage est-il possible pour un problème et des données d'apprentissage particuliers ?
- L'apprentissage est-il réalisable efficacement ?

Les réponses à ces deux questions dépendent en grande partie de l'espace des hypothèses :

- Pour que l'apprentissage soit possible, il est nécessaire qu'il existe un biais d'apprentissage.
- L'efficacité de l'apprentissage dépend de la force de ce biais et de la structuration de l'espace des hypothèses.

Chapitre 2

Première approche théorique de l'induction

Regardons passer un vol de cygnes en file indienne. Pouvons-nous prédire quel sera le prochain oiseau à passer? Encore un cygne? Une oie? Le vol va-t-il s'arrêter? Dans combien de temps passera un autre vol? Que ce soit à l'aide d'expériences de pensée impliquant des volatiles ou plus classiquement des émeraudes (vertes jusqu'ici, mais bleues peut-être à partir de demain¹), les philosophes ont cherché à comprendre l'induction, ce qui rend possible le passage de l'observation d'événements passés à la prédiction.

À ses débuts, l'apprentissage artificiel a effectué de l'induction comme M. Jourdain de la prose, sans s'arrêter à tous les problèmes profonds liés à l'induction comme forme de raisonnement. Après la réalisation de quelques systèmes pionniers et de quelques techniques impressionnantes, mais difficiles à maîtriser, l'étude de l'induction est devenue petit à petit plus rigoureuse.

En apprentissage artificiel, l'induction met en jeu d'abord un problème, donc des règles du jeu et une mesure de performance, ensuite un principe inductif qui spécifie ce qu'est l'hypothèse idéale étant donné un problème, et finalement un algorithme qui réalise au mieux le principe inductif, c'est-à-dire qui cherche effectivement l'optimum défini par le principe inductif. Ce chapitre est concerné par les deux premiers points. Une fois le problème d'apprentissage posé, plusieurs principes inductifs sont imaginables : choisir l'hypothèse qui s'accorde le mieux avec les données observées jusqu'à là, ou bien choisir l'hypothèse permettant la description la plus économique de ces données, ou d'autres encore. L'étude théorique, relativement récente, des conditions de validité de ces grands principes a conduit à la définition de principes inductifs plus sophistiqués qui sont à la base de nouvelles méthodes d'apprentissage.

1. Il s'agit d'une métaphore classique, initialement introduite par le philosophe Goodman, pour discuter de l'induction. Voir le chapitre de J.G Ganascia dans [DKBM00].

TOIT LE MONDE sait distinguer un corbeau d'un canard. La couleur, le cri, la vitesse du vol, la silhouette, beaucoup d'attributs les séparent. Toute personne qui observe l'un de ces deux oiseaux peut lui donner son nom pratiquement sans erreur. Pourtant, cet observateur n'a certainement pas déjà vu *tous* les corbeaux ni *tous* les canards. Mais à partir d'observations en nombre limité, il a appris à les distinguer, c'est-à-dire à trouver des régularités permettant leur identification. Cette forme d'apprentissage, tirant des lois générales à partir d'observations particulières, s'appelle *induction* ou *généralisation*.

Il y a dans le paragraphe ci-dessus un autre exemple d'induction : il est écrit « *tout le monde sait ...* », ce qui est une généralisation (exagérée). Il faudrait d'abord fixer le cadre où cette loi est opératoire, la tempérer par des contraintes géographiques (« *En France, tout le monde...* »), zoologiques (il y a en France beaucoup d'espèces de canards et plusieurs de corbeaux), etc. Mais même dans un cadre plus précis, cette affirmation ne fait que généraliser des observations. Elle signifie en réalité « *presque tous les gens que j'ai observés faire cet exercice sont capables sous certaines conditions de distinguer un corbeau d'un canard* ». D'où la formulation raccourcie, qui énonce une loi extraite d'observations.

Induire : expliquer, prédire, faire simple

Si l'on se place d'un point de vue philosophique, l'induction est liée à plusieurs notions :

- La généralisation, c'est-à-dire le passage d'observations particulières à des classes d'événements ou à des lois s'appuie souvent sur une *recherche d'explications*. En effet, classiquement, une explication scientifique est définie comme une assignation causale. On parvient à expliquer un phénomène si on l'a relié de façon univoque à des antécédents à travers une ou plusieurs loi(s) de la nature. Ainsi un corbeau est différent d'un canard *parce que* la théorie de l'évolution des espèces selon un certain schéma idéalement déductif dicte que, dans nos régions tempérées, en cette période de l'évolution, peut coexister un certain nombre d'espèces présentant certaines caractéristiques spécifiques. Cela détermine des classes d'animaux possibles, dont celles que l'on nomme corbeaux et canards. Si tout va bien, cette théorie va expliquer pourquoi les volatiles peuvent exister dans certaines classes de poids, présenter certaines couleurs, etc. À partir de là, il devient possible de savoir comment distinguer des classes d'oiseaux.
- Une explication, qui consiste à remonter d'un phénomène connu à ses causes inconnues, est valide quand elle peut être retournée en un *outil de prédiction* permettant d'aller de causes connues à des phénomènes encore inconnus. L'induction est donc également liée à la *capacité de prédiction*. Cette prédiction n'est peut-être pas vraie à 100%, mais elle est fondée et généralement valide. Il est extrêmement rare de rencontrer en France des canards noirs, et cela pourrait s'expliquer par la théorie de l'évolution.
- Finalement, à côté de leur pouvoir prédictif, les descriptions et les explications sont aussi jugées à l'aune de leur *simplicité*, de leur élégance et de leur fécondité par rapport à l'ensemble des connaissances. La théorie de l'évolution est-elle bien insérée dans les connaissances générales ? Est-elle performante dans d'autres contextes ? Cette théorie permet-elle de prédire que l'on peut distinguer les canards des corbeaux à partir simplement de leur couleur ? Est-il besoin pour les distinguer de mesurer la longueur des plumes de leurs ailes² ?

L'étude de l'induction est donc liée à celle des concepts d'*explication*, de *prédiction* et d'*économie de description*. Nous trouverons la trace de ces liens tout au long de l'ouvrage.

2. Cet exemple n'est pas pris au hasard : mis à part leur chant, le pouillot véloce (*Phylloscopus Collybita*) et le pouillot fitis (*Phylloscopus Trochilus*) ne se distinguent pratiquement que de cette manière.

L'induction artificielle

Le point de vue décrit ci-dessus présuppose chez l'agent cognitif l'existence de tout un ensemble de connaissances sophistiquées. Ce genre de connaissances complexes (comme la théorie de l'évolution) ne se trouve pas encore chez les agents cognitifs artificiels. Du point de vue de l'ingénieur, le problème est d'inférer par des moyens automatiques une bonne règle de décision à partir d'un échantillon restreint de données sur le phénomène étudié. Cette règle de décision peut avoir deux buts, non contradictoires : soit permettre uniquement la *prédiction* sur une nouvelle observation (l'oiseau que je vois maintenant, est-ce un canard ou un corbeau ?), soit correspondre à la *découverte d'une théorie* générale du phénomène qui à la fois l'explique et permet de prédire ce qui se passera dans chaque cas particulier possible (il suffit de considérer la couleur pour savoir à quel type d'oiseau on a affaire: corbeau ou canard).

D'où les questions fondamentales suivantes :

- Qu'est-ce qui autorise à généraliser à partir d'un échantillon limité de données ?
- Comment réaliser cette extrapolation ? Suivant quel principe ? Est-ce qu'une bonne explication des données disponibles est la promesse d'une bonne capacité de prédiction ?
- Quelles garanties peut-on avoir sur les performances d'une extrapolation ?

Ce chapitre a pour objectif de présenter les outils conceptuels et théoriques qui ont été développés pour répondre à ces questions.

2.1 Deux exemples d'induction

2.1.1 Le système ARCH

Le travail réalisé par Winston pour sa thèse en 1970 [Win70] a servi de précurseur à de nombreuses méthodes d'apprentissage. Le but est d'apprendre des descriptions de concepts à partir d'exemples et de contre-exemples. Le type de concepts étudiés par Winston est celui des arches dans un monde de blocs (voir figure 2.1). Les scènes visuelles correspondantes sont décrites par des réseaux sémantiques tels que celui de la figure 2.2, de même que le concept à apprendre. Ces réseaux permettent à la fois la comparaison directe avec les descriptions des scènes supposées fournies par le système visuel et la représentation de concepts relationnels. Le réseau de la figure 2.2 correspond à peu près aux expressions du calcul des prédictats suivantes :

```
PARTIE_DE(arche,a) ∧ PARTIE_DE(arche,b) ∧ PARTIE_DE(arche,c)
∧ A_LA_PROPRIÉTÉ(a,allongé) ∧ SORTE_DE(a,objet) ∧ DOIT_ETRE_SUPPORTÉ_PAR(a,b)
∧ DOIT_ETRE_SUPPORTÉ_PAR(a,c) ∧ NE_DOIT_PAS_TOUCHER(b,c) ∧ NE_DOIT_PAS_TOUCHER(c,b)
∧ À_GAUCHE_DE(b,c) ∧ À_DROITE_DE(c,b) ∧ A_LA_PROPRIÉTÉ(b,debout)
∧ A_LA_PROPRIÉTÉ(c,debout) ∧ SORTE_DE(b,brigde) ∧ SORTE_DE(c,brigde)
```

avec des faits exprimant une connaissance sur le monde des blocs tels que :

```
SORTE_DE(brigue,objet) ∧ SORTE_DE(debout,propriété)
```

et des faits exprimant des relations entre prédictats, tels que :

```
OPPOSÉ(DOIT_TOUCHER, NE_DOIT_PAS_TOUCHER)
FORME_OBLIGATOIRE(EST_SUPPORTÉ_PAR, DOIT_ÊTRE_SUPPORTÉ_PAR)
```

De cette manière, la représentation choisie permet d'exprimer explicitement des conditions nécessaires. Cela permet d'exprimer en une seule structure des conditions nécessaires et suffisantes.

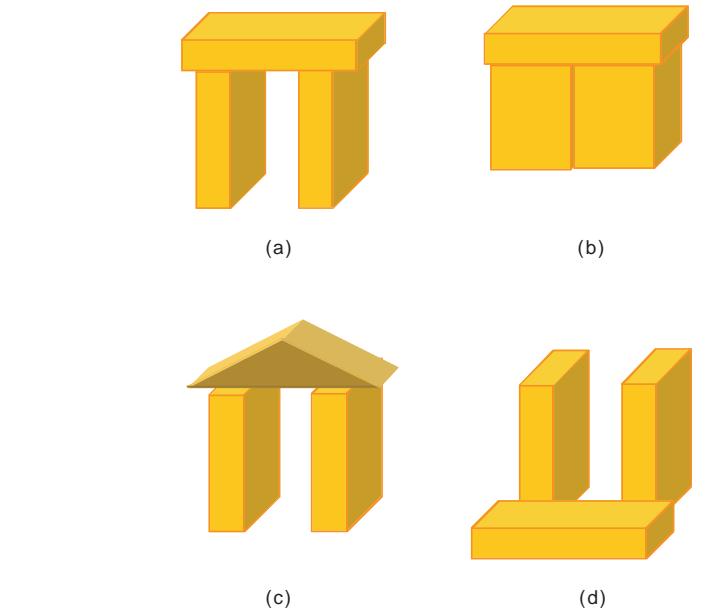


FIG. 2.1 – Dans un monde de blocs, le système ARCH doit apprendre à distinguer les constructions correspondant à des arches (a et c) de celles n'y correspondant pas (b et d).

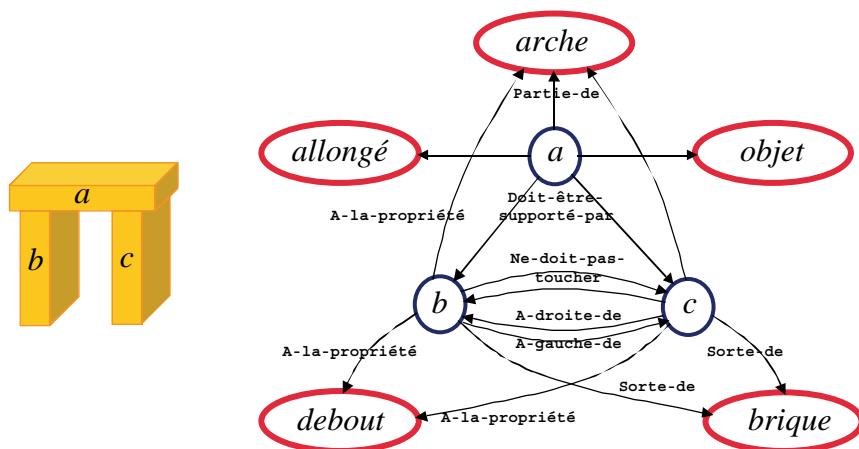


FIG. 2.2 – Un exemple d'arche et sa représentation.

L'algorithme d'apprentissage de Winston prend en compte un par un, séquentiellement, les exemples d'apprentissage (exemples ou contre-exemples du concept à apprendre) et modifie au fur et à mesure son hypothèse courante pour qu'elle couvre, ou encore accepte, le nouvel exemple s'il est positif ou, au contraire, qu'elle ne le couvre pas, le rejette, s'il est négatif. L'idée sous-jacente est que l'hypothèse va ainsi converger vers une hypothèse correspondant au concept cible à apprendre.

L'algorithme est schématiquement le suivant :

- Si le nouvel exemple est *positif*, il faut généraliser l'hypothèse courante pour qu'elle couvre le nouvel exemple si ce n'est pas déjà le cas. L'algorithme ARCH généralise un réseau sémantique soit en éliminant des noeuds ou des liens, soit en remplaçant un noeud (e.g. **cube**) par un concept plus général (e.g. **brique**). Dans certains cas, l'algorithme doit choisir entre ces deux techniques de généralisation. Il essaye d'abord le choix le moins radical (remplacement de noeud) et place l'autre choix sur une pile de possibilités en cas de retour-arrière.
- Si le nouvel exemple est *négatif*, une condition nécessaire (représentée par un lien **doit**) est ajoutée au réseau sémantique représentant l'hypothèse courante. S'il y a plusieurs différences entre l'exemple négatif et l'hypothèse courante, l'algorithme a recours à des règles *ad hoc* pour choisir une différence à « blâmer » expliquant pourquoi l'exemple est négatif. Celle-ci est alors convertie en une condition nécessaire. Les autres différences sont ignorées, mais mises sur une pile pour le cas de retour-arrière.

Cet algorithme suppose qu'il soit aisé de comparer directement l'hypothèse et les exemples d'apprentissage. Ici, le formalisme de représentation est le même et l'on parle alors d'*astuce de représentation unique (single-representation trick)*. Par ailleurs, afin d'éviter l'explosion du nombre de choix à faire par l'algorithme, on suppose que les exemples sont fournis dans un *ordre pédagogique*, de telle manière en particulier que les exemples négatifs ne présentent qu'une seule différence avec l'hypothèse courante. Ce genre d'exemples est appelé *nuance critique (near-miss)*. Cela nécessite bien sûr que l'enseignant connaisse l'hypothèse de l'apprenant à chaque instant. Finalement, il faut remarquer que cet algorithme d'apprentissage est représentatif de deux présupposés communs à quasiment tous les travaux en intelligence artificielle et en psychologie cognitive dans les années 1970.

- Le premier est que le concept cible est supposé provenir d'un professeur et peut donc être parfaitement appris par un apprenant, censé partager le même appareil cognitif et la même représentation des connaissances.
- Le second est que la meilleure hypothèse est celle qui « colle » parfaitement aux données d'apprentissage, c'est-à-dire qui couvre correctement tous les exemples positifs et exclut tous les exemples négatifs.

Nous aurons l'occasion de discuter en profondeur ces deux présupposés dans la suite de ce chapitre.

Le système ARCH est un exemple de système d'*apprentissage inductif symbolique*, c'est-à-dire tirant parti de la représentation des connaissances et des relations entre concepts, y compris de généralité. Nous allons voir maintenant, avec le perceptron, un exemple d'apprentissage non-symbolique ou numérique.

2.1.2 Le perceptron

Le problème auquel s'attaque l'algorithme du perceptron développé par Rosenblatt ([Ros62]) est le suivant : supposons que nous ayons une séquence d'observations $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m$, chacune

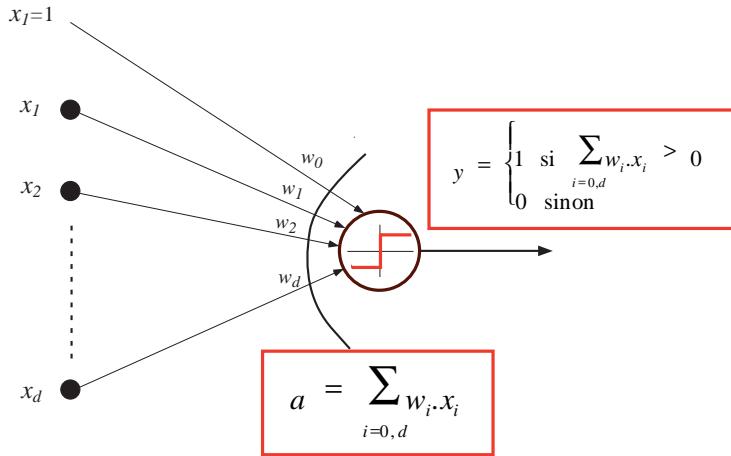


FIG. 2.3 – Le schéma d'un perceptron. À gauche sont figurées les synapses, dont on suppose que le rôle est de pondérer l'entrée correspondante. À droite le signal de sortie est véhiculé par l'axone. Le noyau cellulaire effectue un traitement sur le vecteur d'entrée \mathbf{x} dont chaque composante x_i est pondérée par le poids synaptique correspondant w_i , par exemple en appliquant une fonction seuil $\text{sign}\left(\sum_{i=0}^d w_i x_i\right)$.

d'entre elles étant affectée à une classe $\omega \in \{\omega_1, \omega_2\}$. Par exemple, cela pourrait être des descriptions de volatiles avec comme classe *oie* ou *cygne*. Nous voulons trouver les paramètres d'un automate tel que celui de la figure 2.3 permettant de prédire la classe d'une nouvelle observation.

Cet automate est en fait l'ancêtre des réseaux connexionnistes³. En partant de l'échantillon d'apprentissage, il s'agit donc de trouver un vecteur de poids \mathbf{w} et un seuil w_0 tel que :

$$\mathbf{w}^T \mathbf{x} + w_0 \quad \begin{cases} \geq 0 \\ < 0 \end{cases} \implies \mathbf{x} \in \begin{cases} \omega_1 \\ \omega_2 \end{cases}$$

soit encore :

$$\mathbf{w}^T \mathbf{x} \quad \begin{cases} \geq 0 \\ < 0 \end{cases} \implies \mathbf{x} \in \begin{cases} \omega_1 \\ \omega_2 \end{cases} \quad (2.1)$$

en considérant le vecteur de description des observations augmenté $\mathbf{x}^T = (1, x_1, x_2, \dots, x_d)$ et le vecteur poids \mathbf{w} augmenté du seuil w_0 : $\mathbf{w}^T = (w_0, w_1, w_2, \dots, w_d)$.

Pour ce faire, une méthode inductive s'appuyant sur l'échantillon d'apprentissage consiste à chercher un vecteur \mathbf{w} vérifiant les équations ci-dessus pour tous les exemples d'apprentissage, ou au moins pour le plus grand nombre d'entre eux.

En associant à la classe ω_1 la valeur $u=1$ et à la classe ω_2 la valeur $u=-1$, on cherche un vecteur de poids \mathbf{w} vérifiant :

$$\mathbf{w}^T \mathbf{x} \cdot u > 0 \quad \forall (\mathbf{x}_i, u_i) \in \mathcal{S} \quad (2.2)$$

où \mathcal{S} est l'échantillon d'apprentissage.

3. Ces réseaux sont souvent également appelés « réseaux de neurones » (*neural networks*) dans la littérature. Ceci parce qu'ils sont issus de modélisations de neurones faites dans les années quarante et cinquante. Nous trouvons cependant ce terme abusif et trompeur et nous utiliserons dans cet ouvrage le terme de « réseaux connexionnistes » qui souligne également que les paramètres du système sont les poids des connexions.

Comment apprendre un tel vecteur de poids? Le premier prérequis est de pouvoir mesurer la performance d'un perceptron avec un certain vecteur poids à l'instant t : $\mathbf{w}(t)$. L'idée la plus immédiate est de compter le nombre d'erreurs commises sur l'échantillon d'apprentissage, avec comme objectif de le réduire au maximum. On parle dans ce cas d'*erreur de classification*. Une autre mesure de performance liée à la précédente consiste à calculer un *taux d'erreur en classification* qui est une moyenne de l'erreur en classification rapportée au nombre m d'exemples d'apprentissage. L'inconvénient de ces deux mesures est qu'elles sont discontinues, changeant brusquement pour chaque erreur de classification en plus ou en moins. L'équation 2.2 ci-dessus suggère une autre mesure de performance, connue sous le nom de *critère du perceptron*:

$$R_{Emp}(\mathbf{w}) = - \sum_{x_j \in \mathcal{M}} \mathbf{w}^T \mathbf{x}_j \cdot u_j \quad (2.3)$$

où \mathcal{M} est l'ensemble des exemples d'apprentissage mal classés par le perceptron de vecteur poids \mathbf{w} . Cette mesure d'erreur, que nous appellerons plus tard *risque empirique*, est la somme de termes positifs, est n'est égale à zéro que si tous les exemples d'apprentissage sont bien classés. Cette mesure est proportionnelle à la somme des distances des exemples mal classés à la frontière de décision décrite par l'équation 2.1. Il s'agit donc d'une fonction continue et linéaire par morceaux (et donc non dérivable en un nombre de points égal au nombre d'exemples mal classés). Une fois choisie une mesure de performance, l'un des algorithmes d'apprentissage les plus simples consiste à ajuster le vecteur de poids \mathbf{w} par une procédure de descente de gradient afin d'optimiser la mesure de performance (2.3). On obtient alors une séquence de vecteurs poids $\mathbf{w}(t)$ obéissant à l'équation :

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \frac{\partial R_{Emp}(\mathbf{w}(t))}{\partial \mathbf{w}(t)} \Big|_{\mathbf{w}(t)} \quad (2.4)$$

où η est un réel positif de petite valeur que l'on appelle *pas d'apprentissage*.

Cette procédure d'ajustement doit être répétée en présentant chaque exemple plusieurs fois.

Si cette procédure de descente de gradient est appliquée au critère du perceptron, on obtient l'équation suivante :

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \mathbf{x}_i u_i \quad (2.5)$$

Celle-ci, utilisée avec un critère d'arrêt, définit un algorithme d'apprentissage très simple:

1. Passer en revue chaque exemple dans l'échantillon d'apprentissage et tester la réponse y produite par le perceptron par rapport à la réponse désirée u .
2. Si les deux réponses coïncident – l'exemple est correctement classé –, ne rien faire.
3. Sinon, si l'exemple est incorrectement classé en ω_1 , ajouter le vecteur $\eta \mathbf{x}$ au vecteur poids \mathbf{w} , s'il est incorrectement classé en ω_2 , retirer $\eta \mathbf{x}$ au vecteur poids \mathbf{w} .

Il est alors facile de montrer que l'erreur ne peut que décroître au sens large et même qu'il y a convergence en un nombre fini de présentations d'exemples (voir le chapitre 9 pour une description plus complète du perceptron).

En résumé, ces deux exemples de systèmes très différents ont montré comment il était possible d'apprendre des règles de décision à partir d'exemples étiquetés. Nous avons vu comment un système peut décider de la réponse à faire en présence d'une nouvelle observation, comment on peut mesurer la qualité des décisions prises, quelle forme peut prendre une procédure de

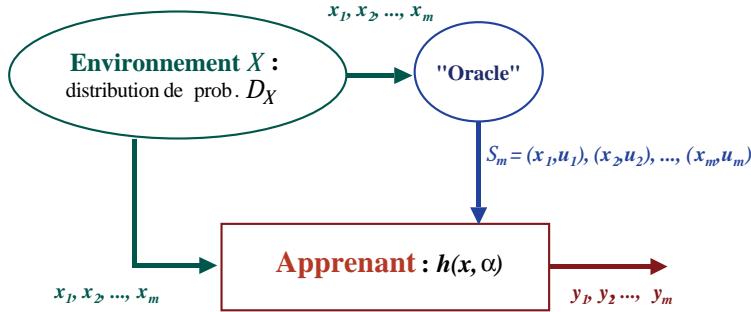


FIG. 2.4 – Le scénario fondamental de l’induction à partir d’un échantillon de données tiré aléatoirement suivant une distribution fixe.

modification de la règle de décision, et enfin, quel type de garantie de convergence on peut chercher sur une procédure d’apprentissage. Nous allons maintenant réexaminer tous ces points d’une manière plus générale.

2.2 Approche de l’induction

Quels sont les facteurs qui entrent en jeu dans l’apprentissage et déterminent le résultat final ? Comment définir un cadre formel permettant de poser le problème de l’induction et d’analyser les méthodes d’induction ? Cette section pose les bases d’une étude plus formelle de l’induction.

2.2.1 Le compromis biais-variance

Le *compromis biais-variance* exprime l’effet de différents facteurs possibles sur l’erreur finale entre l’hypothèse choisie par l’apprenant et celle qu’il *aurait dû* choisir, la *fonction cible* idéale. Nous en donnons ici une description qualitative visant à introduire et à illustrer un certain nombre de concepts utiles pour tout ce chapitre.

Suivant le scénario schématisé dans la figure 2.4, l’apprenant reçoit d’une part un échantillon de données $\{\mathbf{x}_1, \dots, \mathbf{x}_i, \dots, \mathbf{x}_m\}$ composé d’objets ou de *formes* $\mathbf{x}_i \in \mathcal{X}$, où \mathcal{X} est l’*espace de représentation des entrées*. En l’absence d’information supplémentaire sur leur provenance, et pour des raisons de simplicité de modélisation et d’analyse mathématique, on supposera que ces objets sont tirés aléatoirement et indépendamment les uns des autres suivant une distribution de probabilités $\mathcal{D}_{\mathcal{X}}$. C’est ce que l’on appelle l’*hypothèse de tirages indépendants et identiquement distribués* (i.i.d. : *independently and identically distributed*). Attachée à chacune de ces formes \mathbf{x}_i , l’apprenant reçoit d’autre part une *étiquette* ou *supervision* \mathbf{u}_i produite suivant une dépendance fonctionnelle entre \mathcal{X} et \mathcal{U} .

Nous notons $\mathcal{S} = \{\mathbf{z}_1 = (\mathbf{x}_1, \mathbf{u}_1), \dots, \mathbf{z}_m = (\mathbf{x}_m, \mathbf{u}_m)\}$ l’échantillon d’apprentissage constitué ici d’exemples supervisés (ou bien \mathcal{S}_m lorsqu’il sera nécessaire de préciser leur nombre). Pour simplifier, nous supposerons que la dépendance fonctionnelle entre une entrée \mathbf{x}_i et son étiquette \mathbf{u}_i prend la forme d’une fonction f appartenant à une famille de fonctions \mathcal{F} . Nous n’y perdrons pas en généralité si nous supposons également qu’il peut y avoir des informations d’étiquetage erronées, sous la forme d’un *bruit*, c’est-à-dire d’un écart mesurable entre l’étiquette proposée et la véritable étiquette selon f . L’apprenant cherche à trouver une fonction hypothèse h , dans l’espace des fonctions \mathcal{H} , aussi « proche » que possible de f , la fonction cible. Nous préciserons plus tard la notion de proximité utilisée pour évaluer la distance de f à h .

La figure 2.5 illustre les différentes sources d’erreur entre la fonction cible f et la fonction

hypothèse h . Nous appelons *erreur totale* l'erreur résultant de la conjonction de ces différentes erreurs entre f et h . Détaillons-les.

- La première source d'erreur provient du fait suivant: *rien* ne permet *a priori* d'assurer l'égalité entre l'espace des fonctions cible \mathcal{F} de la Nature et l'espace des fonctions hypothèse \mathcal{H} réalisables par l'apprenant. De ce fait, même si l'apprenant fournit une hypothèse h^* optimale (au sens de la mesure de la proximité évoquée plus haut), h^* est forcément prise dans \mathcal{H} et peut donc être différente de la fonction cible f . Il s'agit là d'une *erreur d'approximation* souvent appelée *biais inductif* (ou simplement *biais*⁴) due à la différence entre \mathcal{F} et \mathcal{H} .
- Ensuite, l'apprenant ne fournit en général pas h^* , l'hypothèse optimale dans \mathcal{H} , mais calcule une hypothèse \hat{h} sur la base de l'échantillon d'apprentissage \mathcal{S} . En fonction de cet échantillon, l'hypothèse \hat{h} apprise pourra varier à l'intérieur d'un ensemble de fonctions que nous noterons $\{\hat{h}\}_{\mathcal{S}}$ pour souligner la dépendance de chacun de ses éléments sur l'échantillon aléatoire \mathcal{S} . La distance entre h^* et l'hypothèse estimée \hat{h} , qui dépend des particularités de \mathcal{S} , est l'*erreur d'estimation*. On peut montrer formellement qu'elle est une *variance* liée à la sensibilité du calcul de l'hypothèse \hat{h} en fonction de l'échantillon \mathcal{S} . Plus l'espace d'hypothèses \mathcal{H} est riche et plus, en général, cette variance est importante.
- Finalement, intervient le *bruit* sur l'étiquetage : à cause d'erreurs de transmission, l'étiquette u_i associée à x_i peut être inexacte vis-à-vis de f . De ce fait, l'apprenant reçoit un échantillon de données relatif à la *fonction bruitée* $f_b = f + \text{bruit}$. Il s'agit là d'une *erreur intrinsèque* qui complique généralement la recherche de l'hypothèse optimale h^* .

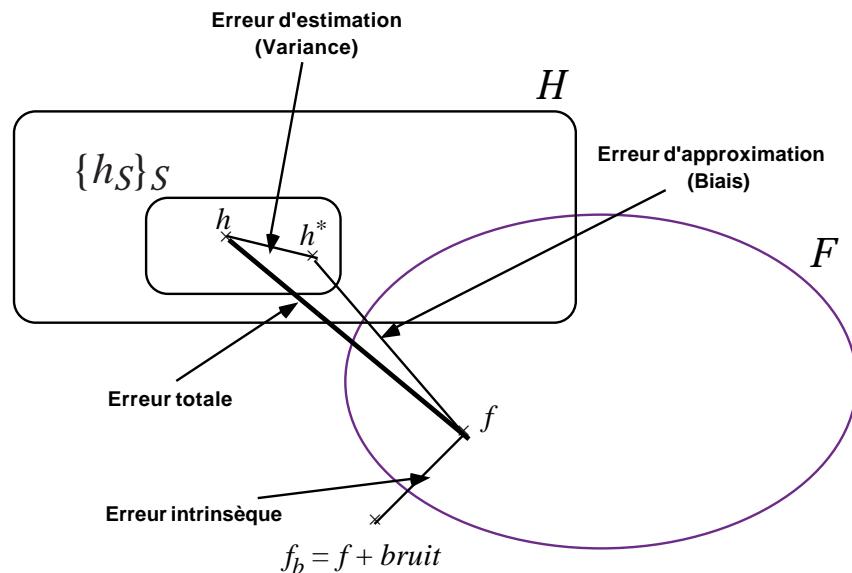


FIG. 2.5 – Les différents types d'erreurs intervenant dans l'estimation d'une fonction cible à partir d'un échantillon d'apprentissage. Avec un espace d'hypothèses plus restreint, comme \mathcal{H}' , on peut réduire la variance, mais généralement au prix d'une plus grande erreur d'approximation.

4. Nous reviendrons souvent sur la notion de biais en apprentissage artificiel. Elle est de fait plus générale que le biais inductif qui exprime les limites de l'espace des hypothèses à décrire le monde représenté dans \mathcal{X} .

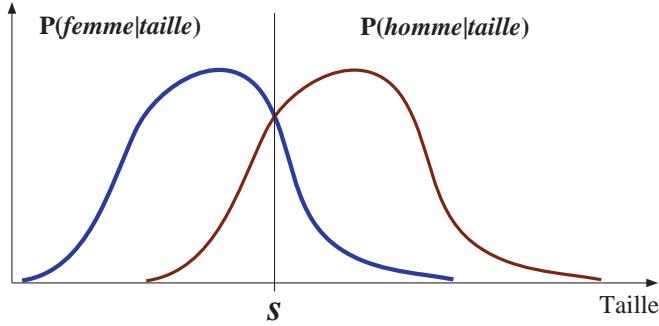


FIG. 2.6 – Soient les deux courbes de probabilité des femmes et des hommes en fonction de la taille. On peut chercher à déterminer si un individu donné appartient à l'une des classes ou à l'autre en comparant la taille mesurée à un seuil s déterminé expérimentalement à l'aide d'un échantillon de personnes des deux sexes.

Étant données ces circonstances, le *compromis biais-variance* se définit de la façon suivante : pour réduire le biais dû à la mauvaise adéquation de \mathcal{H} par rapport à \mathcal{F} , il faut accroître la richesse de \mathcal{H} . Malheureusement, cet enrichissement de \mathcal{H} va généralement de pair avec une augmentation de la variance, c'est-à-dire de l'erreur de la recherche de l'élément optimal *dans* \mathcal{H} . De ce fait, l'*erreur totale*, qui est la somme de l'erreur d'approximation et de l'erreur d'estimation, ne peut être significativement diminuée.

Le compromis biais-variance devrait donc plutôt s'appeler *compromis erreur d'approximation/erreur d'estimation*. Mais l'important est qu'il s'agit bien de faire un compromis, puisqu'on joue sur une somme de termes qui varient ensemble en sens contraire. En revanche le bruit, ou *erreur intrinsèque*, ne peut qu'aggraver les choses en augmentant. L'idéal serait d'avoir un bruit nul (mais comment s'en assurer ?) et d'avoir un espace d'hypothèses \mathcal{H} restreint pour réduire la variance, mais en même temps *bien informé*, c'est-à-dire contenant seulement des fonctions proches de la fonction cible, ce qui reviendrait évidemment à disposer d'une connaissance *a priori* sur la Nature.

Exemple 3 (Exemple de compromis biais-variance)

À titre d'illustration, revenons sur l'exemple du paragraphe 2.1.2. L'espace \mathcal{H} des hypothèses possibles y est celui des *hyperplans*. Soit alors le problème suivant : les « objets » décrits sont des êtres humains et l'espace de représentation est constitué d'un seul attribut : la taille. Le problème d'apprentissage consiste à discriminer les hommes des femmes. Dans ce cas, un hyperplan se réduit à un seuil s sur le seul attribut disponible, la taille, et l'apprentissage consiste à le déterminer à partir des exemples. Pour savoir si un individu est un homme ou une femme, on mesure sa taille et on la compare au seuil s . Le biais, ou erreur d'approximation, est ici important puisque l'espace des fonctions choisi est pauvre et ne permet pas une discrimination parfaite entre hommes et femmes. Cela signifie que la décision prise sur le seul critère d'un seuil s sur la taille fera de nombreuses erreurs. En revanche, la variance, ou erreur d'estimation, peut être rendue pratiquement nulle : même en prenant des échantillons différents tirés aléatoirement dans une population donnée, les courbes $P(\text{femme}|\text{taille})$ et $P(\text{homme}|\text{taille})$ seront stables et par conséquent le seuil s calculé sera de variance faible.

Supposons maintenant qu'au lieu de mesurer seulement la taille, on prenne des dizaines de mesures, disons cent, sur chacun des individus exemples : par exemple, leur taille, la couleur de leur peau, la longueur de leur cheveux, etc. Le biais sera baissé, voire nul, car il est imaginable

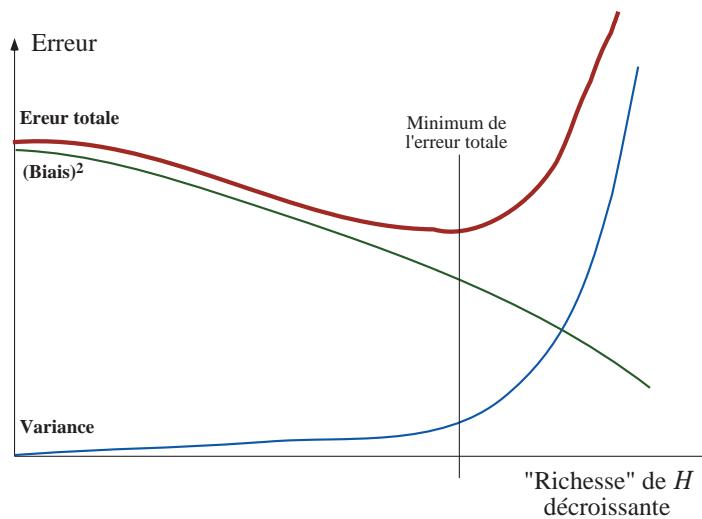


FIG. 2.7 – Il est possible d'essayer de diminuer l'erreur totale accessible en réglant la richesse de l'espace d'hypothèses \mathcal{H} .

qu'il existe un hyperplan discriminant les hommes des femmes dans un espace de représentation d'une telle richesse. En revanche, il est beaucoup plus difficile d'estimer cent paramètres qu'un seul⁵ : on risque donc, sur la base d'un échantillon de données forcément limité, de ne pas trouver le meilleur hyperplan (erreur d'estimation). Une autre illustration du même problème est que, pour le même nombre m d'exemples, la précision des estimations sera très douteuse. On aura donc cette fois une forte variance sur les hypothèses produites à partir de différents échantillons d'apprentissage.

Le compromis biais-variance peut s'illustrer par la figure 2.7 dans laquelle l'erreur totale est tracée comme la somme du biais⁶ et de la variance. Au fur et à mesure que l'on diminue la richesse de \mathcal{H} le biais augmente et la variance diminue. L'erreur totale atteint en général un minimum pour un certain *réglage* du biais. On voit là une indication de ce qu'il est possible de chercher à faire pour améliorer l'apprentissage. Nous verrons des réalisations de cette idée dans la section 2.6 décrivant les grands principes inductifs servant de base aux algorithmes d'apprentissage.

2.2.2 Comment définir formellement le problème de l'induction ?

Après avoir ainsi fourni les éléments pour une compréhension intuitive du problème, nous en donnons maintenant une expression plus formelle.

Un problème d'apprentissage est défini par les composantes suivantes :

1. D'abord un ensemble de trois acteurs :

- *L'environnement*: il est supposé stationnaire et il engendre des formes \mathbf{x}_i tirées indépendamment et de manière identiquement distribuées (échantillon i.i.d.) suivant une distribution $\mathcal{D}_{\mathcal{X}}$ sur l'espace d'entrée \mathcal{X} .
- Un *oracle* ou *superviseur* ou *professeur* ou *Nature*, qui, pour chaque forme \mathbf{x}_i retourne

5. En reconnaissance des formes, ce problème est connu sous le nom de *la malédiction de la dimensionnalité* (*curse of dimensionality*)

6. En réalité, pour des raisons d'homogénéité mathématique, on emploie généralement le carré du biais.

une réponse désirée ou étiquette \mathbf{u}_i en accord avec une distribution de probabilité conditionnelle $F(\mathbf{u}|\mathbf{x})$ inconnue.

- Un apprenant \mathcal{A} capable de réaliser une fonction (non nécessairement déterministe) prise dans un espace de fonctions \mathcal{H} , telle que la sortie produite par l'apprenant vérifie : $\mathbf{y}_i = h(\mathbf{x}_i)$ pour $h \in \mathcal{H}$.
2. Ensuite, la *tâche d'apprentissage* : l'apprenant cherche dans l'espace \mathcal{H} une fonction h qui approxime au mieux la réponse désirée de l'oracle. Dans le cas de l'induction, la *distance* entre la fonction hypothèse h et la réponse de l'oracle est définie par l'*espérance de perte* sur les situations possibles dans $\mathcal{Z} = \mathcal{X} \times \mathcal{U}$. Ainsi, pour chaque entrée \mathbf{x}_i et réponse de l'oracle \mathbf{u}_i , on mesure une *perte* ou *coût* (*loss* en anglais) $l(\mathbf{u}_i, h(\mathbf{x}_i))$ évaluant le coût d'avoir pris la décision $\mathbf{y}_i = h(\mathbf{x}_i)$ quand la réponse désirée était \mathbf{u}_i . (On supposera, sans perte de généralité, ce coût toujours positif ou nul). L'espérance de coût, ou *risque réel* est alors :

$$R_{\text{Réel}}(h) = \int_{\mathcal{Z}=\mathcal{X} \times \mathcal{U}} l(\mathbf{u}_i, h(\mathbf{x}_i)) dF(\mathbf{x}, \mathbf{u}) \quad (2.6)$$

Il s'agit d'une mesure statistique qui est fonction de la dépendance fonctionnelle $F(\mathbf{x}, \mathbf{u})$ entre les entrées \mathbf{x} et les sorties désirées \mathbf{u} . Cette dépendance peut être exprimée par une densité de probabilité conjointe définie sur $\mathcal{X} \times \mathcal{U}$ qui est inconnue. En d'autres termes, il s'agit de trouver une hypothèse h proche de f au sens de la fonction de perte, et ce particulièrement dans les régions de l'espace des entrées \mathcal{X} fréquemment rencontrées. Comme on ne connaît pas *a priori* ces régions, il faut utiliser l'échantillon d'apprentissage pour les estimer, et le problème de l'induction est donc de chercher à minimiser le risque réel inconnu à partir de l'observation de l'échantillon d'apprentissage \mathcal{S} .

3. Finalement, un *principe inductif* qui prescrit ce que doit vérifier la fonction h recherchée, en fonction à la fois de la notion de proximité évoquée ci-dessus et de l'échantillon d'apprentissage observé $\mathcal{S} = \{(\mathbf{x}_1, \mathbf{u}_1), (\mathbf{x}_2, \mathbf{u}_2), \dots, (\mathbf{x}_m, \mathbf{u}_m)\}$, dans le but de minimiser le risque réel.

Le *principe inductif* dicte ce que doit vérifier la meilleure hypothèse en fonction de l'échantillon d'apprentissage, de la fonction de perte et, éventuellement, d'autres critères. Il s'agit d'un objectif idéal. Il faut le distinguer de la *méthode d'apprentissage* (ou algorithme) qui décrit une réalisation effective du principe inductif. Pour un principe inductif donné, il y a de nombreuses méthodes d'apprentissage qui résultent de choix différents pour régler les problèmes computationnels qui ne sont pas du ressort du principe inductif. Par exemple, le principe inductif peut prescrire qu'il faut choisir l'hypothèse la plus simple compatible avec l'échantillon d'apprentissage. La méthode d'apprentissage doit alors spécifier comment chercher effectivement cette hypothèse, ou une hypothèse suboptimale s'il y a lieu, en satisfaisant certaines contraintes de réalisabilité comme des ressources computationnelles. Ainsi, par exemple, la méthode d'apprentissage cherchera par une méthode de gradient, sub-optimale mais facilement contrôlable, l'optimum défini par le principe inductif.

La définition donnée ci-dessus est très générale : en particulier, elle ne dépend pas de la fonction de perte choisie (voir annexe 18.1 pour les principales fonctions de perte). Elle a le mérite de distinguer les principaux ingrédients d'un problème d'apprentissage qui sont souvent mélangés dans les descriptions de réalisations pratiques.

2.2.3 Quel principe inductif adopter ? Une introduction

Le principe inductif prescrit quelle hypothèse on devrait choisir pour minimiser le risque réel sur la base de l'observation d'un échantillon d'apprentissage. Or il n'y a pas de principe inductif

unique ou idéal. Comment extraire, à partir des données, une régularité qui ait des chances d'avoir une pertinence pour l'avenir? Un certain nombre de réponses « raisonnables » ont été proposées. Nous en décrivons les principales de manière qualitative ici avant de les réexaminer plus formellement dans la suite de ce chapitre ainsi que dans le chapitre 17.

1. **Le choix de l'hypothèse minimisant le risque empirique** (*Empirical Risk Minimization* en anglais ou principe *ERM*) . Le risque empirique est la perte moyenne mesurée sur l'échantillon d'apprentissage \mathcal{S} :

$$R_{emp}(h) = \frac{1}{m} \sum_{i=1}^m l(\mathbf{u}_i, h(\mathbf{x}_i)) \quad (2.7)$$

L'idée sous-jacente à ce principe est que l'hypothèse qui s'accorde le mieux aux données, en supposant que celles-ci soient représentatives, est une hypothèse qui décrit correctement le monde en général. Le principe de minimisation du risque empirique a été, souvent implicitement, le principe utilisé en intelligence artificielle depuis l'origine, tant dans le connexionnisme que dans l'apprentissage symbolique. Quoi de plus naturel en effet que de considérer qu'une régularité observée sur les données connues sera encore vérifiée par le phénomène qui a produit ces données? C'est par exemple le principe directeur de l'algorithme du perceptron comme de celui du système ARCH. Dans ces deux cas, on cherche une hypothèse cohérente avec les exemples, c'est-à-dire de risque empirique nul.

Nous examinerons plus loin (chapitre 4) le cas où il existe une relation d'ordre de généralité entre les hypothèses. Il sera alors possible d'affiner le principe de minimisation du risque empirique en choisissant parmi les hypothèses optimales, soit l'une des plus spécifiques, soit l'une des plus générales.

2. **Le choix de l'hypothèse la plus probable étant donné l'échantillon d'apprentissage.** C'est le *principe de décision bayésienne*. L'idée est ici qu'il est possible de définir une distribution de probabilité sur l'espace des fonctions hypothèse et que la connaissance du domaine préalable à l'apprentissage peut en particulier s'exprimer sous la forme d'une distribution de probabilité *a priori* sur les hypothèses. L'échantillon d'apprentissage est alors considéré comme une information modifiant la distribution de probabilité sur \mathcal{H} (voir la figure 2.8). On peut alors, soit choisir l'hypothèse la plus probable *a posteriori* (*principe du maximum de vraisemblance*) ou *Maximum A Posteriori (MAP)*, soit adopter une hypothèse composite résultant de la moyenne des hypothèses pondérée par leur probabilité *a posteriori* (*vraie approche bayésienne*).

Pour prendre un exemple dramatique tiré de l'actualité de l'année 2001, lorsque Rudolph Giuliani, alors maire de New-York, s'est réveillé le matin du 11 septembre 2001, la probabilité d'un avion percutant accidentellement un gratte-ciel était jugée très faible *a priori*, et plus encore celle d'un avion en percutant un intentionnellement. C'est pourquoi, après la nouvelle du premier impact sur l'une des Twin Towers, le maire s'est-il demandé comment un avion avait-il pu s'égarer hors des couloirs aériens. Il s'agissait alors de l'hypothèse la plus probable pour expliquer la collision. Après le second impact sur l'autre tour, l'hypothèse d'un accident devenait négligeable devant celle d'un acte délibéré. Deux exemples suffisaient ici à modifier de manière considérable les probabilités *a priori* sur l'espace des hypothèses.

3. **Le choix d'une hypothèse qui comprime au mieux les informations contenues dans l'échantillon d'apprentissage.** Nous appellerons ce précepte: *principe de compression d'information*. L'idée est d'éliminer les redondances présentes dans les données afin

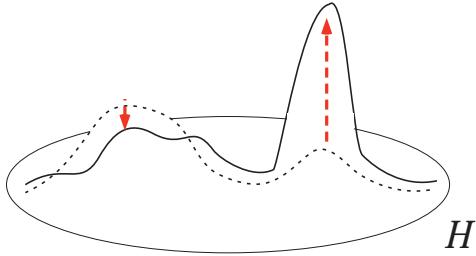


FIG. 2.8 – L'espace des hypothèses \mathcal{H} est supposé muni d'une densité de probabilités a priori. L'apprentissage consiste à modifier cette densité en fonction des exemples d'apprentissage.

d'extraire les régularités sous-jacentes permettant la description économique du monde. Il est sous-entendu que les régularités découvertes dans les données sont valides au-delà des données et s'appliquent au monde en général.

Il faut noter que ces principes inductifs qui servent de cadres (parfois implicites) aux travaux sur l'apprentissage artificiel, ne sont pas incompatibles entre eux, même s'ils relèvent de philosophies et, souvent, de communautés scientifiques différentes.

La question est de savoir si ces idées intuitivement séduisantes permettent d'apprendre *effectivement*. Plus précisément, nous aimerais obtenir des réponses à un certain nombre de questions « naïves » :

- L'application du principe inductif sélectionné conduit-elle à minimiser effectivement le risque réel ?
- Faut-il pour cela vérifier certaines conditions ? Par exemple sur l'échantillon d'apprentissage, ou bien sûr les fonctions cibles, ou bien sur l'oracle, ou encore sur l'espace des fonctions hypothèses.
- Comment joue l'échantillon d'apprentissage sur la performance en généralisation, c'est-à-dire sur le risque réel ?
- Plus précisément encore, comment la performance en généralisation dépend-elle de l'information contenue dans l'échantillon d'apprentissage, ou bien de sa taille, etc. ?
- Quelle performance maximale est possible pour un problème d'apprentissage donné ?
- Quel est le meilleur apprenant pour un problème d'apprentissage donné ?

Répondre à ces questions implique des choix qui dépendent en partie du type de principe inductif utilisé. C'est pourquoi nous en avons fait une description succincte ci-dessus.

2.2.4 Comment analyser l'apprentissage ?

Nous avons décrit l'apprentissage, du moins l'apprentissage inductif, comme un problème d'optimisation : chercher la meilleure hypothèse au sens de la minimisation de l'espérance de risque sur la base d'un échantillon d'apprentissage. Nous voulons maintenant étudier sous quelles conditions la résolution d'un tel problème est possible. Nous voulons aussi avoir des outils permettant de juger de la performance d'un principe inductif ou d'un algorithme d'apprentissage. Cette analyse requiert des hypothèses supplémentaires qui correspondent à des options sur ce qui est attendu du système d'apprentissage.

Ainsi, un problème d'apprentissage dépend de l'environnement qui soumet des formes d'apprentissage x_i suivant une certaine distribution inconnue \mathcal{D}_X , de l'oracle qui choisit une fonction

cible f et de la fonction de perte choisie l . La performance d'un apprenant (qui dépend du principe inductif choisi et de l'algorithme d'apprentissage le réalisant) sera évaluée en fonction des choix de chacun de ces paramètres. Lorsque nous cherchons à déterminer la performance *attendue* d'un apprenant, nous devons donc prendre position sur la source de ces paramètres. Trois postures en particulier sont possibles :

1. On suppose que l'on ne connaît rien *a priori* sur l'environnement, donc ni sur la distribution des formes d'apprentissage, ni sur la dépendance cible, mais l'on veut se prémunir contre la pire des situations possibles, comme si l'environnement et l'oracle étaient des adversaires. On cherche alors à caractériser la performance de l'apprenant dans les pires situations envisageables, ce qui se traduit généralement par des bornes. C'est *l'analyse dans le pire cas*. On parle aussi de cadre d'analyse MinMax, par référence à la théorie des jeux. L'avantage de ce point de vue est que les garanties de performances éventuelles seront indépendantes de l'environnement (le risque réel étant calculé quelle que soit la distribution des événements) et de l'oracle ou de la Nature (c'est-à-dire quelle que soit la fonction cible). En revanche, les conditions identifiées pour obtenir de telles garanties seront tellement fortes qu'elles seront souvent très éloignées des situations d'apprentissage réelles.
2. On peut au contraire vouloir mesurer une espérance de performance. Dans ce cas, il faut supposer qu'il existe une distribution $\mathcal{D}_{\mathcal{X}}$ sur les formes d'apprentissage, mais aussi une distribution $\mathcal{D}_{\mathcal{F}}$ sur les fonctions cible possibles. L'analyse qui en résulte est une *analyse en cas moyen*. On parle aussi de cadre bayésien. Cette analyse permet en principe une caractérisation plus fine de la performance, au prix cependant de devoir faire des hypothèses *a priori* sur les états du monde. Malheureusement, il est souvent très difficile d'obtenir analytiquement des conditions de garanties d'apprentissage réussi, et il faut généralement utiliser des méthodes d'approximation qui enlèvent une partie de l'intérêt d'une telle approche.
3. Finalement, on pourrait chercher à caractériser le cas le plus favorable, lorsque l'environnement et l'oracle sont *bienveillants* et veulent aider l'apprenant. Mais il est difficile de déterminer la frontière entre la bienveillance, celle d'un professeur par exemple, et la *collusion* qui verrait alors l'oracle agir comme un complice et coder la fonction cible dans un code connu de l'apprenant, ce qui ne serait plus de l'apprentissage, mais une transmission illicite. C'est pourquoi ce type d'analyse, quoique intéressant, n'a pas encore de cadre bien établi.

2.3 Analyse dans le pire cas : l'apprentissage *PAC*

Dans cette section, nous nous concentrerons sur l'analyse du principe inductif *ERM* qui prescrit de choisir une hypothèse minimisant le risque empirique mesuré sur l'échantillon d'apprentissage. C'est en effet la règle la plus employée, et son analyse conduit à des principes conceptuels très généraux. Le principe *ERM* a d'abord fait l'objet d'une analyse dans le pire cas, que nous décrivons ici. Une analyse en cas moyen, faisant intervenir des idées de physique statistique, a également fait l'objet de nombreux travaux très intéressants. Elle est cependant techniquement nettement plus difficile et est discutée dans le chapitre 17 portant sur des approfondissements théoriques de l'étude de l'apprentissage.

2.3.1 Étude des conditions de validité de l'ERM

Rappelons que l'apprentissage consiste à chercher une hypothèse h telle qu'elle minimise l'espérance de perte pour l'apprenant. Formellement, il s'agit de trouver une hypothèse optimale h^* minimisant le risque réel :

$$h^* = \underset{h \in \mathcal{H}}{\text{ArgMin}} R_{\text{Réel}}(h) \quad (2.8)$$

Le problème est que l'on ne connaît pas le risque réel attaché à chaque hypothèse h . L'idée naturelle est donc de sélectionner une hypothèse h dans \mathcal{H} qui se comporte bien sûr les données d'apprentissage \mathcal{S} : c'est le principe inductif de la *minimisation du risque empirique (ERM)*. Nous noterons $\hat{h}_{\mathcal{S}}$ cette hypothèse optimale pour le risque empirique mesuré sur l'échantillon \mathcal{S} :

$$\hat{h}_{\mathcal{S}} = \underset{h \in \mathcal{H}}{\text{ArgMin}} R_{\text{Emp}}(h) \quad (2.9)$$

Ce principe inductif ne sera pertinent que si le risque empirique est corrélé avec le risque réel. Son analyse doit donc s'attacher à étudier la **corrélation entre les deux risques** et plus particulièrement la corrélation entre le risque réel encouru avec l'hypothèse sélectionnée à l'aide du principe *ERM*: $R_{\text{Emp}}(\hat{h}_{\mathcal{S}})$ et le risque réel optimal : $R_{\text{Réel}}(h^*)$.

Cette corrélation va faire intervenir deux aspects :

1. La **différence** (forcément positive ou nulle) **entre le risque réel de l'hypothèse sélectionnée à partir de l'échantillon d'apprentissage** $\hat{h}_{\mathcal{S}}$ et le risque réel de l'hypothèse optimale h^* : $R_{\text{Réel}}(\hat{h}_{\mathcal{S}}) - R_{\text{Réel}}(h^*)$
2. **La probabilité que cette différence soit supérieure à une borne donnée** ε .

Étant donné en effet que le risque empirique dépend de l'échantillon d'apprentissage, la corrélation entre le risque empirique mesuré et le risque réel dépend de la *représentativité* de cet échantillon. Si celle-ci est mauvaise, l'échantillon d'apprentissage peut conduire, via le principe *ERM*, à choisir une mauvaise hypothèse. Par exemple, il n'est pas impossible qu'avec une pièce non truquée on tire cent fois « pile ». On ne pourra alors reprocher à quelqu'un d'avoir, sur la base d'un tel échantillon, parié que la pièce est truquée. Fort heureusement, la probabilité que cela arrive est faible. C'est pourquoi aussi, lorsque l'on étudie la différence $R_{\text{Réel}}(h^*) - R_{\text{Réel}}(\hat{h}_{\mathcal{S}})$, il faut prendre en compte la probabilité de l'échantillon d'apprentissage étant donnée une certaine fonction cible. On ne peut pas être un bon apprenant à tous les coups, mais seulement dans les situations raisonnables (échantillons d'apprentissage représentatifs) qui sont les plus probables.

Reprendons donc la question de la corrélation entre risque empirique et risque réel. Le principe de minimisation du risque empirique est un principe inductif valide si, lorsque l'on choisit l'hypothèse $\hat{h}_{\mathcal{S}}$ qui minimise le risque empirique, le risque réel qui en résulte est garanti d'être proche du risque réel optimal obtenu avec l'hypothèse optimale h^* , et ceci dans la grande majorité des situations qui peuvent se présenter, c'est-à-dire pour la majorité des échantillons d'apprentissage tirés aléatoirement suivant la distribution $\mathcal{D}_{\mathcal{X}}$.

De manière plus formelle, on cherche sous quelles conditions il serait possible d'assurer :

$$\forall 0 \leq \varepsilon, \delta \leq 1 : P(|R_{\text{Réel}}(\hat{h}_{\mathcal{S}}) - R_{\text{Réel}}(h^*)| \geq \varepsilon) < \delta \quad (2.10)$$

La figure 2.9 illustre ce problème. Nous cherchons à déterminer sous quelles conditions la probabilité que l'on choisisse une mauvaise hypothèse (dont le risque réel est supérieur au risque réel optimal de plus de ε) est bornée par δ .

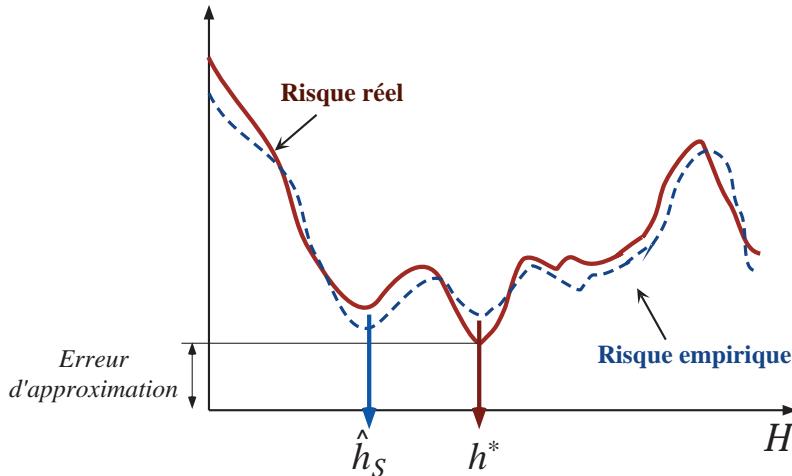


FIG. 2.9 – La courbe continue indique le risque réel associé à chaque hypothèse $h \in \mathcal{H}$ placé ici en abscisse. La courbe en pointillés indique le risque empirique associé à ces hypothèses pour un échantillon d'apprentissage \mathcal{S} . Le principe de minimisation du risque empirique (ERM) dicte de choisir l'hypothèse minimisant ce risque empirique. Nous voudrions borner la probabilité que, ce faisant, le risque réel encouru soit à plus de ε du risque réel optimal.

On comprend bien que la corrélation entre le risque empirique et le risque réel dépend de l'échantillon d'apprentissage \mathcal{S} , et, puisque celui-ci est tiré aléatoirement (tirage i.i.d.), de sa taille m . Cela suggère naturellement une application de la *loi des grands nombres* selon laquelle, sous des conditions très générales, la moyenne d'une variable aléatoire (e.g. $R_{Emp}(h)$) converge vers son espérance (ici $R_{Réel}(h)$) lorsque croît la taille m de l'échantillon.

La loi des grands nombres incite à vouloir assurer l'équation (2.10) en faisant croître la taille de l'échantillon d'apprentissage \mathcal{S} vers ∞ et à se demander à partir de quelle taille m d'un échantillon d'apprentissage tiré aléatoirement (tirage i.i.d. suivant une distribution $\mathcal{D}_{\mathcal{X}}$ quelconque), l'équation (2.10) est garantie :

$$\forall 0 \leq \varepsilon, \delta \leq 1 : \exists m, \text{ tq. } P(|R_{Réel}(h^*) - R_{Réel}(\hat{h}_{\mathcal{S}_m})| \geq \varepsilon) < \delta \quad (2.11)$$

La courbe (2.10) illustre la convergence souhaitée du risque empirique vers le risque réel.

Définition 2.1 (Pertinence du principe ERM)

On dit que le principe ERM est pertinent⁷ si le risque réel inconnu $R_{Réel}(\hat{h}_{\mathcal{S}})$ et le risque empirique $R_{Emp}(\hat{h}_{\mathcal{S}})$ convergent vers la même limite $R_{Réel}(h^*)$ lorsque la taille m de l'échantillon

7. « Consistance » est le terme généralement employé, repris directement de l'anglais. Dans le cas présent, ce terme n'évoque rien en français et fait un double emploi malheureux avec le terme d'apprenant « consistant » employé parfois pour décrire un apprenant qui cherche une hypothèse dont les prédictions sur tous les exemples de l'échantillon d'apprentissage sont en accord avec les réponses fournies par l'oracle. Le terme d'apprenant consistant est aussi employé pour décrire un apprenant dont l'erreur réelle tend vers l'erreur optimale de Bayes lorsque la taille de l'échantillon m tend vers l'infini (voir [DEV96]). Nous préférons introduire un nouveau terme, celui de *pertinence* qui traduit bien ce que l'on cherche : la validité du principe inductif *ERM*.

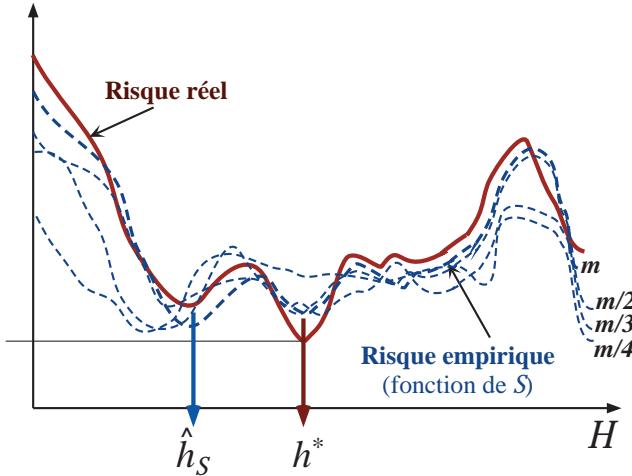


FIG. 2.10 – La courbe continue indique le risque réel associé à chaque hypothèse $h \in \mathcal{H}$ placé ici en abscisse. Les courbes en pointillés indiquent le risque empirique associé à ces hypothèses pour un échantillon d'apprentissage \mathcal{S}_m . Si le principe inductif ERM est pertinent, alors ces courbes doivent converger vers la courbe de risque réel lorsque la taille m de l'échantillon d'apprentissage croît.

tend vers ∞ (voir la figure 2.11).

$$\begin{aligned} R_{Réel}(\hat{h}_{\mathcal{S}}) &\xrightarrow{m \rightarrow \infty} R_{Réel}(h^*) \\ R_{Emp}(\hat{h}_{\mathcal{S}}) &\xrightarrow{m \rightarrow \infty} R_{Réel}(h^*) \end{aligned} \quad (2.12)$$

Dans ce cas en effet, il est justifié d'utiliser le principe inductif *ERM* pour choisir une hypothèse à partir de l'observation d'un échantillon de données.

Malheureusement, la loi des grands nombres n'est pas suffisante pour notre étude. En effet, ce que cette loi affirme, c'est que le risque empirique d'une hypothèse donnée h converge vers son risque réel. Mais ce que nous cherchons est différent. Nous voulons assurer que l'hypothèse $\hat{h}_{\mathcal{S}_m}$, prise dans \mathcal{H} et qui minimise le risque empirique pour l'échantillon \mathcal{S} , a un risque réel associé qui converge vers le risque réel optimal obtenu pour l'hypothèse optimale h^* , indépendante de \mathcal{S} . Il faut bien voir que dans ce cas l'échantillon d'apprentissage ne joue plus uniquement le rôle d'un ensemble de test, mais aussi d'un ensemble servant au choix de l'hypothèse. On ne peut donc plus prendre sans précaution la performance mesurée sur l'échantillon d'apprentissage comme représentatrice de la performance réelle.

Ainsi par exemple, on peut supposer que les notes obtenues par un étudiant lors d'examens répétés convergent vers sa vraie valeur sur la matière testée. Cependant, si on choisit dans la classe l'étudiant ayant obtenu la meilleure moyenne sur ces examens, on n'est pas garantit d'avoir ainsi l'étudiant le plus performant réellement. Il se peut en effet que le vrai meilleur étudiant n'ait pas été le meilleur sur les examens donnés. Il s'agit ici d'une convergence plus complexe, dans laquelle doivent intervenir le nombre et la diversité des étudiants.

Un autre exemple donné par les théoriciens est qu'on peut construire des espaces d'hypothèses \mathcal{H} tels qu'il soit toujours possible de trouver une hypothèse de risque empirique nul sans que cela indique une bonne performance générale, donc sans qu'il y ait corrélation entre le risque empirique minimal et le risque réel minimal. Il suffit pour cela d'imaginer une hypothèse qui s'accorde à toutes les données d'apprentissage et qui tire au hasard l'étiquette des formes non

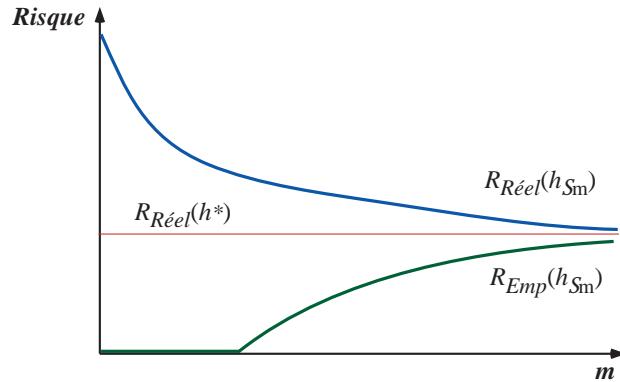


FIG. 2.11 – Pertinence du principe ERM. À cause du biais ou erreur d'approximation, le risque réel optimal atteint pour l'hypothèse h^* n'est généralement pas égal à zéro. Au fur et à mesure que la taille m de l'échantillon croît, le risque réel associé à l'hypothèse courante sélectionnée \hat{h}_{S_m} diminue et tend vers le risque optimal. Le risque empirique est inférieur au risque réel puisque \hat{h}_{S_m} est sélectionnée pour minimiser le premier et non le second. Souvent, il arrive que pour de petits échantillons de données, on trouve une hypothèse dans \mathcal{H} dont le risque empirique est nul. Cela devient généralement impossible lorsque la taille m de l'échantillon croît.

vues. C'est pourquoi il faut généraliser la loi des grands nombres.

Cette généralisation est aisée dans le cas d'un espace fini de fonctions hypothèses, elle n'a été obtenue que récemment par Vapnik et Chervonenkis (1971,1989), dans le cadre de l'induction, pour le cas des espaces de dimension infinie. Pour des raisons pédagogiques, nous n'étudions dans ce chapitre que le cas de l'apprentissage de fonctions indicatrices (à valeur dans {0,1}) prises dans un espace d'hypothèses fini. Le cas général de fonctions hypothèse quelconques est traité dans le chapitre 17.

2.3.2 Le cas de la discrimination : l'analyse PAC

Dans la lignée des travaux en intelligence artificielle symbolique (dont le système ARCH est un prototype), les premiers pas en théorie de l'apprenabilité ont été initiés par Valiant (1984), focalisés sur les tâches d'apprentissage de concepts, c'est-à-dire de fonctions binaires ou encore *indicatrices* sur l'espace \mathcal{X} des exemples. Il était de plus supposé que la fonction cible f appartenait à l'espace des hypothèses \mathcal{H} , c'est-à-dire que l'apprenant pouvait apprendre parfaitement la fonction cible⁸. Dans le cas, que nous supposerons vérifié ici, où l'échantillon d'apprentissage n'est pas bruité, il est donc toujours possible de trouver une hypothèse de risque empirique nul. Cette hypothèse répond donc « 1 » sur tous les exemples positifs et « 0 » sur tous les exemples négatifs. C'est ce que nous avons appelé une hypothèse cohérente avec l'échantillon d'apprentissage.

L'apprentissage peut alors être considéré comme un processus éliminatoire qui écarte toutes les hypothèses non cohérentes avec l'échantillon d'apprentissage, et en choisit une parmi les restantes.

8. Dans le contexte des sciences cognitives privilégié en Occident dans les années quatre-vingt, on supposait en effet implicitement que l'apprenant et l'oracle étaient tous les deux des êtres humains et donc qu'ils partageaient le même appareil cognitif. L'apprenant était alors vu comme cherchant à apprendre parfaitement le concept cible du « maître » qui lui était par nature accessible.

Rien ne garantit que l'hypothèse ainsi obtenue soit exactement la fonction cible: plusieurs hypothèses peuvent en effet être de risque empirique nul sur l'échantillon d'apprentissage. La proximité de l'hypothèse choisie avec la fonction cible dépend donc de cet échantillon. C'est pourquoi on a appelé les travaux théoriques portant sur ce type d'apprentissage *analyse PAC* [Ang88] pour *apprentissage Probablement Approximativement Correct*.

Nous commençons par une analyse d'un cas simple dans lequel l'espace des fonctions cibles est fini. Nous cherchons alors à borner l'erreur commise lorsque, en accord avec le principe *ERM*, on choisit une hypothèse cohérente avec les données d'apprentissage.

2.3.2.1 Le cas où $|\mathcal{H}|$ est fini

Nous supposons une fonction cible f fixée, ainsi que la distribution $\mathcal{D}_{\mathcal{X}}$ des exemples dans \mathcal{X} . Voulant évaluer le principe *ERM*, nous cherchons quelle est la probabilité qu'une hypothèse de risque empirique nul sur l'échantillon d'apprentissage soit en fait de risque réel supérieur ou égal à ε , avec $0 \leq \varepsilon \leq 1$.

Nous supposons ici que la fonction de perte l compte le nombre d'erreurs de classification⁹. (voir l'annexe 18.1) :

$$l(\mathbf{u}_i, h(\mathbf{x}_i)) = \begin{cases} 0 & \text{si } \mathbf{u}_i = h(\mathbf{x}_i) \\ 1 & \text{si } \mathbf{u}_i \neq h(\mathbf{x}_i) \end{cases} \quad (2.13)$$

Dans ce cas, le risque réel est égal à la probabilité qu'un exemple tombe dans la zone d'erreur entre la fonction cible f et la fonction hypothèse erronée h_{err} : $h_{err}\Delta f$ (le symbole Δ dénote la différence symétrique entre deux ensembles. Voir figure 2.12).

$$R_{Réel}(h_{err}) = P_{\mathcal{D}_{\mathcal{X}}}(h_{err}\Delta f) \quad (2.14)$$

Posons par hypothèse que l'écart entre f et h_{err} est supérieur à une constante :

$$\varepsilon P_{\mathcal{D}_{\mathcal{X}}}(h_{err}\Delta f) > \varepsilon$$

La probabilité qu'après l'observation d'un exemple on ne s'aperçoive pas que h_{err} est erronée est de $1 - \varepsilon$. Après l'observation d'un échantillon i.i.d. suivant la distribution $\mathcal{D}_{\mathcal{X}}$ de m exemples, la probabilité de « survie » de h_{err} vaut donc $(1 - \varepsilon)^m$.

En considérant maintenant l'ensemble \mathcal{H} des hypothèses possibles, la probabilité que l'une d'entre elles « survive » après l'observation de \mathcal{S} est bornée par : $|\mathcal{H}|(1 - \varepsilon)^m$ (on fait ici une sommation car on a affaire à une union d'événements disjoints). On sait, par développement limité, que $|\mathcal{H}|(1 - \varepsilon)^m < |\mathcal{H}|e^{-\varepsilon m}$. En reprenant l'inéquation 2.11, il suffit donc d'avoir un échantillon de taille m telle que :

$$m \geq \frac{1}{\varepsilon} \ln \frac{|\mathcal{H}|}{\delta} \quad (2.15)$$

pour que l'erreur commise en choisissant l'hypothèse $\hat{h}_{\mathcal{S}}$ minimisant le risque empirique soit bornée par ε avec une probabilité $> 1 - \delta$.

On retiendra de cette démonstration trois idées :

1. D'abord que la cardinalité de \mathcal{H} , donc en un certain sens sa richesse, a un effet direct sur la borne d'erreur. Il est déjà apparent que le choix d'un ensemble \mathcal{H} trop riche peut conduire à de mauvaises inductions.
9. Attention !! Même si ce que nous allons dire dans cette section peut se généraliser à d'autres fonctions de perte, le détail des démonstrations dépend de cette hypothèse, et ne peut donc se transposer sans précautions à d'autres fonctions de perte.

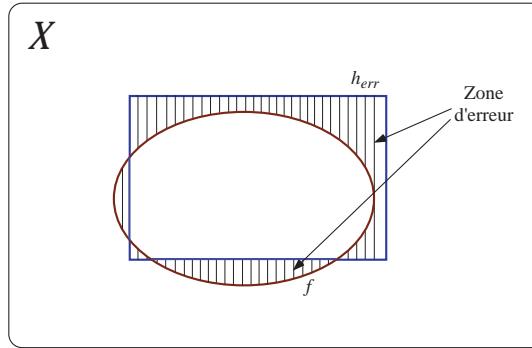


FIG. 2.12 – La zone d'erreur dans le cas de l'apprentissage d'un concept ou fonction binaire définie sur \mathcal{X} .

2. Ensuite, le raisonnement utilisé dans la démonstration implique l'*ensemble* des fonctions hypothèse de \mathcal{H} . Nous verrons qu'une généralisation de ce raisonnement fait appel de même à un argument de *convergence uniforme*. Cette observation est très importante car elle indique que l'analyse est *de facto* une analyse dans le pire cas, s'appliquant en particulier à l'hypothèse pour laquelle la convergence est la plus mauvaise.
3. Finalement, l'idée directrice de la démonstration consiste à borner la probabilité qu'une zone d'erreur de poids $> \varepsilon$ ne soit pas atteinte par un exemple de l'échantillon d'apprentissage¹⁰.

Remarque

Le cas où la fonction cible $f \notin \mathcal{H}$.

Pour une hypothèse donnée $h \in \mathcal{H}$, une forme particulière de la loi des grands nombres, l'*inégalité de Hoeffding* [Hoe56, Hoe63a], donne la vitesse avec laquelle la « queue » de la distribution binomiale approche zéro. Elle s'applique à la convergence du risque empirique sur le risque réel pour une hypothèse h :

$$P(|R_{Réel}(h^*) - R_{Réel}(\hat{h}_{\mathcal{S}_m})| \geq \varepsilon) < 2 e^{-2\varepsilon^2 m} \quad (2.16)$$

pour toute distribution $\mathcal{D}_{\mathcal{X}}$, tout ε et tout entier positif m . Il faut retenir qu'en première approximation, la convergence de la fréquence d'erreur sur la probabilité d'erreur est en $1/\sqrt{m}$, ce qui correspond au type de convergence associée au théorème central limite.

Nous voulons borner l'erreur commise par l'apprenant lorsqu'il choisit l'hypothèse $\hat{h}_{\mathcal{S}} \in \mathcal{H}$, dépendant de l'échantillon d'apprentissage \mathcal{S} , au lieu de l'hypothèse optimale h^* . Pour cela, il suffit de montrer que la différence entre le risque empirique et le risque réel de n'importe quelle hypothèse est bornée. La figure 2.13 montre pourquoi. En effet, si l'écart entre le risque empirique et le risque réel de n'importe quelle hypothèse h est borné par ε , alors l'écart entre le risque réel associé à $\hat{h}_{\mathcal{S}}$ et le risque réel optimal associé à h^* est borné par 2ε . Cela se traduit par un théorème:

Théorème 2.1

Soit \mathcal{H} un ensemble fini de fonctions indicatrices (définies de \mathcal{X} vers $\{0,1\}$). Alors :

$$P\left(\max_{h \in \mathcal{H}} |R_{Réel}(h) - R_{Emp}(h)| \geq \varepsilon\right) < 2 |\mathcal{H}| e^{-2\varepsilon^2 m} \quad (2.17)$$

pour toute distribution $\mathcal{D}_{\mathcal{X}}$, tout ε et tout entier positif m .

10. Une généralisation de cette idée a été utilisée dans le cas d'espaces de fonctions indicatrices de cardinalité infinie avec la notion de ε -réseau (ε -net)[Hau92].

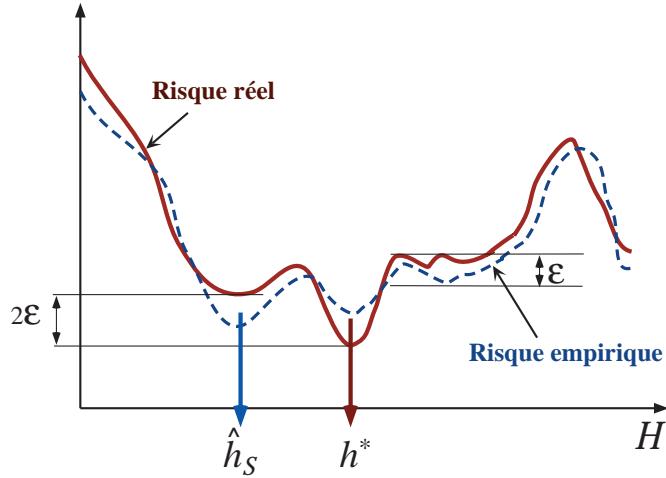


FIG. 2.13 – Si chaque hypothèse $h \in \mathcal{H}$ a un risque empirique proche de son risque réel (à moins de ε), alors minimiser le risque empirique (en application du principe inductif ERM) minimisera approximativement aussi le risque réel (à au plus 2ε), ce qui est le but recherché et assure la pertinence du principe ERM.

Démonstration. Cela résulte du fait que la probabilité d'une union d'événements indépendants est inférieure ou égale à la somme de leurs probabilités. \square

Nous avons là à nouveau un exemple d'un argument de *convergence uniforme*: il montre que le risque empirique converge vers le risque réel lorsque m tend vers ∞ , uniformément pour toutes les hypothèses de \mathcal{H} .

Dans ce cas, il suffit d'avoir un échantillon d'apprentissage \mathcal{S}_m de taille :

$$m \geq \frac{2}{\varepsilon^2} \ln \frac{2|\mathcal{H}|}{\delta} \quad (2.18)$$

pour borner par ε avec une probabilité $> 1 - \delta$ l'erreur commise en choisissant l'hypothèse $\hat{h}_{\mathcal{S}}$ de risque empirique minimal au lieu de l'hypothèse optimale.

Démonstration. En effet, le théorème 2.1 affirme que :

$$P(\max_{h \in \mathcal{H}} |R_{Réel}(h) - R_{Emp}(h)| \geq \varepsilon) < 2|\mathcal{H}| e^{-2\varepsilon^2 m}$$

et cela ne vaut pas plus que δ si :

$$\varepsilon \geq \left(\frac{1}{2m} \ln \frac{2|\mathcal{H}|}{\delta} \right)^{1/2}$$

Alors, avec une probabilité $\geq 1 - \delta$, $\forall h \in \mathcal{H}$:

$$R_{Réel}(h) - \varepsilon < R_{Emp}(h) < R_{Réel}(h) + \varepsilon$$

d'où :

$$\begin{aligned} R_{Réel}(\hat{h}_{\mathcal{S}}) &\leq R_{Emp}(\hat{h}_{\mathcal{S}}) + \varepsilon \\ &\leq R_{Emp}(h^*) + \varepsilon \quad \text{Puisque par application du principe ERM: } \hat{h}_{\mathcal{S}} = \operatorname{ArgMin}_{h \in \mathcal{H}} R_{Emp}(h) \\ &< (R_{Réel}(h^*) + \varepsilon) + \varepsilon \\ &= R_{Réel}(h^*) + 2\varepsilon \end{aligned}$$

Donc, avec probabilité $\geq 1-\delta$: $R_{R\acute{e}el}(\hat{h}_{\mathcal{S}}) \leq R_{R\acute{e}el}(h^*) + \left\{ \frac{2}{m} \ln \frac{2|\mathcal{H}|}{\delta} \right\}^{1/2}$ soit, en posant $\varepsilon = \left\{ \frac{2}{m} \ln \frac{2|\mathcal{H}|}{\delta} \right\}^{1/2}$:

$$m \leq \frac{2}{\varepsilon^2} \ln \frac{2|\mathcal{H}|}{\delta} \quad (2.19)$$

□

On remarquera que cette borne inférieure sur l'échantillon d'apprentissage est bien moins bonne que la borne obtenue en (2.15) lorsque la fonction cible f est supposée appartenir à l'ensemble des fonctions hypothèses \mathcal{H} : il y a maintenant un facteur ε^2 au dénominateur au lieu d'un facteur ε . Pourquoi ?

Une justification intuitive peut être avancée. Dans le cas où la fonction cible $f \in \mathcal{H}$, cela signifie que la distribution des exemples sur $\mathcal{X} \times \{0,1\}$, et la partition induite sur \mathcal{X} , peuvent être « représentées » par une fonction de \mathcal{H} . Il s'agit donc d'un sous-ensemble des distributions possibles sur $\mathcal{X} \times \{0,1\}$. De ce fait, la variance sur les hypothèses $\hat{h}_{\mathcal{S}}$ qui minimisent le risque empirique en fonction de l'échantillon d'apprentissage \mathcal{S} est réduite. Or moins de données sont nécessaires pour approcher une variable aléatoire dont la variance est moindre. Nous aurons l'occasion de revenir sur cette différence entre le cas parfaitement apprenable (erreur d'estimation nulle) et le cas apprenable par approximation. Retenons qu'il est remarquablement plus facile de chercher une bonne hypothèse dans le cas apprenable. Il est tentant dans ces conditions de faire bon usage de cette remarque fondamentale.

Sans avoir rendu compte de l'analyse beaucoup plus complète de Vapnik (décrise dans le chapitre 17), nous pouvons retenir à ce stade que le principe inductif de minimisation du risque empirique ne peut être appliqué sans précaution. Pour que la mesure du risque empirique soit corrélée avec le risque réel, il faut que l'espace d'hypothèses \mathcal{H} dans lequel on choisit \hat{h} ait de bonnes propriétés. De manière informelle, il faut que cet espace ne soit pas trop « riche » ou trop « souple », c'est-à-dire qu'on ne puisse pas y trouver des hypothèses s'accordant à n'importe quel jeu de données. On retrouve naturellement une idée déjà rencontrée avec le compromis biais-variance. Cela signifie que le principe *ERM* doit être modifié pour que la richesse de \mathcal{H} soit également prise en compte lorsque l'on recherche la meilleure hypothèse. Toutes les techniques de contrôle de l'espace d'hypothèses visent à régler ce compromis. Nous les examinerons plus loin.

2.4 Analyse dans un cas moyen : l'analyse bayésienne

Il est évidemment d'un intérêt essentiel pour l'étude du problème de l'induction de savoir s'il existe une *manière optimale* d'utiliser l'information disponible dans l'échantillon d'apprentissage. Si tel était le cas, cela fournirait une *borne inférieure* pour le risque réel qui ne pourrait être battue en moyenne par aucun algorithme d'apprentissage. La question deviendrait alors de savoir à combien de cet optimum un principe inductif peut s'approcher et s'il existe un algorithme d'apprentissage qui puisse effectivement atteindre cette borne. Ces questions font l'objet de cette section. La *règle de décision bayésienne* y est en effet définie comme étant la règle de décision optimale au sens où elle minimise le risque réel en utilisant l'information disponible de manière optimale.

On peut objecter que, dans le cas de la classification par exemple, tout ce que l'échantillon d'apprentissage apporte comme information est que la fonction cible figure dans l'ensemble des fonctions cohérentes avec les exemples d'apprentissage. Comment dès lors dépasser le principe *ERM* et spécifier un apprenant optimal ? Il est en effet nécessaire de se donner des informations supplémentaires sur la probabilité *a priori* des fonctions cible. Il sera alors possible de définir une règle de décision minimisant l'espérance de risque. C'est ce que nous allons voir maintenant.

2.4.1 Nature de l'analyse bayésienne

L'analyse de l'induction dans le pire cas étudie les conditions sous lesquelles il existe de bonnes garanties, quantifiables, pour que quelle que soit la dépendance cible dans la nature, quelle que soit la distribution des événements mesurables, toute hypothèse qui colle bien aux données d'apprentissage (en minimisant le risque empirique) soit effectivement proche (au sens du risque réel) de la dépendance cible. Il est clair que cette formulation du problème de l'induction fait bien place à la question centrale du rapport entre ce qui a été observé et le futur.

L'analyse bayésienne se pose le problème de l'induction dans des termes différents. D'un certain côté, il s'agit d'une question beaucoup plus pratique: *étant donné que l'on m'a fourni un échantillon de données, comment cela doit-il modifier mes croyances antérieures sur le monde?* Cette formulation introduit immédiatement deux aspects:

- D'une part, quelle forme prend la connaissance ou ensemble de croyances sur le monde?
- D'autre part, quelle règle normative doit présider à la révision de la connaissance en présence de nouvelles données?

Le révérend Thomas Bayes (1702-1761) a répondu à ces deux questions. Il propose d'une part que la connaissance sur le monde soit traduite par un ensemble d'hypothèses (fini ou non), chacune d'entre elles étant affectée d'une probabilité reflétant le degré de croyance de l'apprenant dans l'hypothèse en question. La connaissance sur le monde est ainsi exprimée sous la forme d'une *distribution de probabilités sur un espace d'hypothèses*. D'autre part, il donne une *règle de révision* permettant de modifier une distribution *a priori*, en une distribution *a posteriori* tenant compte des données d'apprentissage. En gros, cette règle dit que la probabilité *a posteriori* d'une hypothèse est égale à sa probabilité *a priori* multipliée par la vraisemblance des données étant donnée l'hypothèse. Plus formellement, la célèbre *règle de Bayes* de révision des probabilités s'écrit :

$$p_{\mathcal{H}}(h|\mathcal{S}) = \frac{p_{\mathcal{H}}(h) P_{\mathcal{X}}^m(\mathcal{S}|h)}{P_{\mathcal{X}}(\mathcal{S})} \quad (2.20)$$

où nous dénotons par $p_{\mathcal{H}}$ la densité de probabilité définie sur l'espace des hypothèses $h \in \mathcal{H}$, par $P_{\mathcal{X}}$ la mesure de probabilité des événements sur \mathcal{X} et par $P_{\mathcal{X}}^m$ la mesure de probabilité d'un ensemble d'apprentissage $\mathcal{S} = ((\mathbf{x}_1, \mathbf{u}_1), (\mathbf{x}_2, \mathbf{u}_2), \dots, (\mathbf{x}_m, \mathbf{u}_m))$. Avec ces notations, $p_{\mathcal{H}}(h)$ et $p_{\mathcal{H}}(h|\mathcal{S})$ dénotent respectivement la densité de probabilité *a priori* de l'hypothèse $h \in \mathcal{H}$ et sa densité de probabilité *a posteriori* après prise en compte des données \mathcal{S} . $P_{\mathcal{X}}(\mathcal{S}|h)$ est la probabilité conditionnelle de l'événement \mathcal{S} si l'on suppose vrai l'état du monde correspondant à h . $P_{\mathcal{X}}(\mathcal{S})$ est la probabilité *a priori* de l'événement \mathcal{S} . On pourrait ainsi avoir :

$$\begin{aligned} P_{\mathcal{H}}(\text{oiseau} = \text{canard} | \text{couleur-aile} = \text{noir}) &= \\ \frac{P_{\mathcal{H}}(\text{oiseau} = \text{canard}) p_{\mathcal{X}}(\text{la couleur de l'aile est sombre} | \text{oiseau} = \text{canard})}{p_{\mathcal{X}}(\text{la couleur de l'aile est sombre})} \end{aligned}$$

L'importance pratique de la règle de Bayes tient au fait qu'elle permet de réexprimer la probabilité *a posteriori* difficile à calculer, en termes de probabilités *a priori* et conditionnelles plus faciles à obtenir. Nous y reviendrons.

On peut noter que la règle de Bayes est à la fois normative, dictant ce qui doit être vérifié pour que l'induction, c'est-à-dire la révision des connaissances, soit correcte, et prescriptive, donnant une procédure à suivre pour ce faire. Cela n'est cependant pas suffisant pour spécifier quelle décision il faut prendre en présence de données d'apprentissage. C'est l'objet de la *théorie bayésienne de la décision*.

2.4.2 Le risque bayésien et la décision optimale

Comme dans le cas du principe inductif *ERM*, le but de l'agent est de prendre une décision (par exemple: faire un diagnostic, reconnaître un projet à soutenir, décider si le signal du sonar correspond à une mine entre deux eaux ou à un banc de poissons, etc.) minimisant l'espérance de risque.

Nous définissons une fonction de décision $\mathcal{S} : \mathcal{X} \rightarrow \mathcal{H}$, où \mathcal{H} est ici vu comme un ensemble de décisions à prendre, cet ensemble n'étant pas forcément isomorphe à l'ensemble des états possibles du monde. Par exemple, une décision peut être une décision de *rejet*, c'est-à-dire de ne pas prendre de décision parce que l'incertitude est trop forte et ne permet pas de se décider avec suffisamment de garantie pour une hypothèse sur le monde plutôt qu'une autre.

Avant toute observation sur le monde, et en prenant seulement en compte les connaissances *a priori*, l'*espérance de risque* associée à une décision h peut s'écrire :

$$R(h) = \sum_{f \in \mathcal{F}} l(h|f) p_{\mathcal{F}}(f) \quad (2.21)$$

où $p_{\mathcal{F}}(f)$ dénote la probabilité *a priori* que le monde soit dans l'état f , tandis que $l(h|f)$ est le *coût* ou *perte* encouru lorsque la décision h est prise alors que l'état du monde est f .

On supposera ici tous les coûts positifs ou nuls: $l(h|f) \geq 0, \forall h \in \mathcal{H}, \forall f \in \mathcal{F}$. En général, le coût d'une décision correcte est pris comme nul et celui d'une décision de rejet comme constant: $l(\text{rejet}|f) = r, \forall f \in \mathcal{F}$. Si tout coût de décision incorrecte est équivalent, on aura alors la *fonction de coût* suivante :

$$l(h|f) = \begin{cases} 0 & \text{si } h = f \text{ (décision correcte)} \\ 1 & \text{si } h \neq f \text{ (décision incorrecte)} \\ r & \text{si } h = \text{rejet} \text{ (doute trop important)} \end{cases} \quad (2.22)$$

Dans de nombreuses applications cependant, le coût de décision incorrect dépend à la fois de la décision h et de l'état du monde f et n'est pas de surcroît symétrique. Par exemple, le coût de ne pas diagnostiquer à tort une tumeur est souvent bien plus élevé que de faire un faux diagnostic positif.

Dans le cas de l'équation 2.21, la décision optimale h^* à prendre est évidemment celle correspondant au risque minimal.

Définition 2.2 (Règle de décision bayésienne)

On appelle règle de décision bayésienne la règle de choix de l'hypothèse minimisant l'espérance de risque.

$$h^* = \operatorname{ArgMin}_{h \in \mathcal{H}} R(h) = \operatorname{ArgMin}_{h \in \mathcal{H}} l(h|f) p_{\mathcal{F}}(f) \quad (2.23)$$

En tenant compte du coût de la décision de rejet, cette règle de décision devient :

$$h^* = \begin{cases} \operatorname{ArgMin}_{h \in \mathcal{H}} l(h|f) p_{\mathcal{F}}(f) & \text{si } \min_{h \in \mathcal{H}} R(h) < r \\ \text{rejet} & \text{sinon} \end{cases} \quad (2.24)$$

Évidemment, elle ne présente guère d'intérêt en l'absence de mesures ou nouvelles données dont l'agent peut disposer sur le monde (tant que je n'ai pas un texte sous les yeux et des formes à interpréter comme étant des lettres, je dois faire le pari que j'ai affaire à des 'E', puisque c'est

la lettre la plus probable en français). Lorsque que de nouvelles données \mathcal{S} sont disponibles, il faut utiliser la formule de Bayes de révision des probabilités pour obtenir la règle bayésienne de décision optimale. L'espérance de risque attachée à l'hypothèse h étant donnée l'observation \mathcal{S} est :

$$R(h|\mathcal{S}) = \int_{f \in \mathcal{F}} l(h|f) p_{\mathcal{F}}(f|\mathcal{S}) \quad (2.25)$$

ou bien :

$$R(h|\mathcal{S}) = \sum_{f \in \mathcal{F}} l(h|f) P_{\mathcal{F}}(f|\mathcal{S})$$

dans le cas d'un nombre fini de décisions f possibles, et la règle de décision bayésienne de risque minimal stipule de choisir l'hypothèse h qui minimise ce risque :

$$\begin{aligned} h^* &= \operatorname{ArgMin}_{h \in \mathcal{H}} R(h|\mathcal{S}) = \operatorname{ArgMin}_{h \in \mathcal{H}} \int_{f \in \mathcal{F}} l(h|f) p_{\mathcal{F}}(f|\mathcal{S}) \\ &= \operatorname{ArgMin}_{h \in \mathcal{H}} \int_{f \in \mathcal{F}} l(h|f) \frac{p_{\mathcal{F}}(f) p_{\mathcal{X}}^m(\mathcal{S}|f)}{p_{\mathcal{X}}^m(\mathcal{S})} \end{aligned} \quad (2.26)$$

Comme $p_{\mathcal{X}}(\mathcal{S})$ est un terme indépendant de l'hypothèse considérée, il n'intervient que comme un facteur de normalisation et n'influence pas la décision. On obtient alors :

Définition 2.3 (Règle de décision bayésienne de risque minimal)

La règle de décision bayésienne minimisant l'espérance de risque est :

$$h^* = \operatorname{ArgMin}_{h \in \mathcal{H}} \int_{f \in \mathcal{F}} l(h|f) p_{\mathcal{F}}(f) p_{\mathcal{X}}^m(\mathcal{S}|f) \quad (2.27)$$

Définition 2.4 (Risque de Bayes)

Le risque associé à l'hypothèse optimale h^ est appelé risque de Bayes. Il représente le risque minimal atteignable si l'on connaît la distribution de probabilité a priori sur les états du monde \mathcal{F} ainsi que les probabilités conditionnelles $p_{\mathcal{X}}(\mathcal{S}|f)$.*

En tenant compte de la possibilité de rejet, la règle de décision devient :

$$h^* = \begin{cases} \operatorname{ArgMin}_{h \in \mathcal{H}} \int_{f \in \mathcal{F}} l(h|f) p_{\mathcal{F}}(f) p_{\mathcal{X}}^m(\mathcal{S}|f) & \text{si } \min_{h \in \mathcal{H}} R(h) < r \\ \text{rejet} & \text{sinon} \end{cases} \quad (2.28)$$

Afin d'illustrer l'analyse bayésienne, nous étudions deux cas simples dans la suite, qui correspondent aussi à des règles de décision très classiques.

2.4.3 Cas particuliers de la décision bayésienne

2.4.3.1 Cas de coût de classification incorrecte uniforme: la règle d'erreur minimale

Lorsque le domaine est insuffisamment connu ou bien qu'il se prête à cette hypothèse, on suppose que les coûts de classification incorrecte sont tous équivalents. Par exemple, dans le cas

de la reconnaissance de caractères, on pourrait supposer que le coût de prendre une lettre pour une autre est le même, quelles que soient les lettres en question. On a alors la fonction de coût :

$$l(h|f) = \begin{cases} 0 & \text{si } h = h_f \text{ (décision correcte)} \\ 1 & \text{si } h \neq f \text{ (décision incorrecte)} \end{cases} \quad (2.29)$$

Dans ce cas, la règle de décision bayésienne devient :

$$\begin{aligned} h^* &= \operatorname{ArgMin}_{h \in \mathcal{H}} \sum_{f \in \mathcal{F}} l(h|h_f) p_{\mathcal{F}}(f) p_{\mathcal{X}}(\mathbf{x}|f) \\ &= \operatorname{ArgMin}_{h \in \mathcal{H}} \left\{ l(h|h_f) p_{\mathcal{F}}(h) p_{\mathcal{X}}(\mathbf{x}|h) + \sum_{h \neq h_f} l(h|h_f) p_{\mathcal{F}}(f) p_{\mathcal{X}}(\mathbf{x}|f) \right\} \\ &= \operatorname{ArgMin}_{h \in \mathcal{H}} \left\{ \sum_{h_f \neq h} p_{\mathcal{F}}(f) p_{\mathcal{X}}(\mathbf{x}|f) \right\} \quad (\text{car } l(h|h) = 0 \text{ et } l(h|h_f) = 1) \\ &= \operatorname{ArgMin}_{h \in \mathcal{H}} \{1 - [p_{\mathcal{F}}(f) p_{\mathcal{X}}(\mathbf{x}|f)]\} \\ &= \operatorname{ArgMax}_{h \in \mathcal{F}} p_{\mathcal{F}}(f) p_{\mathcal{X}}(\mathbf{x}|f) \end{aligned}$$

Définition 2.5 (Règle de Maximum *a posteriori* (MAP))

Lorsque les coûts de mauvaise classification sont égaux, la règle de décision bayésienne de risque minimal devient la règle de Maximum A Posteriori: (MAP).

$$h^* = \operatorname{ArgMax}_{h \in \mathcal{F}} p_{\mathcal{F}}(f) p_{\mathcal{X}}(\mathbf{x}|f) \quad (2.30)$$

On cherche en effet l'hypothèse h la plus probable étant donnée l'observation \mathbf{x} , c'est-à-dire a posteriori. Cette règle s'appelle aussi la règle d'erreur minimale car elle minimise le nombre d'erreurs de classification.

Définition 2.6 (Le classificateur bayésien naïf (« naive bayesian classifier »))

Si l'on suppose que les attributs de description $\{a_1, \dots, a_d\}$ de l'espace d'entrée \mathcal{X} sont indépendants les uns des autres, alors on peut décomposer $p_{\mathcal{X}}(\mathbf{x}|f)$ en $p(a_1 = v_{1\mathbf{x}}|f) \dots p(a_d = v_{d\mathbf{x}}|f)$ soit $\prod_{i=1}^d p(a_i = v_{i\mathbf{x}}|f)$. Le classifieur utilisant la règle du Maximum a posteriori basé sur ces hypothèses est appelé classifieur bayésien naïf.

Il faut noter que les attributs de description sont rarement indépendants les uns des autres (par exemple le *poids* et la *taille*). Pourtant le classifieur bayésien naïf donne souvent des résultats proches de ceux obtenus par les meilleures méthodes connues. Domingos et Pazzani, par exemple, dans [DP97] étudient pourquoi.

Définition 2.7 (Règle du maximum de vraisemblance)

Si de plus toutes les hypothèses ont la même probabilité a priori, alors la règle de Maximum A Posteriori devient la règle du Maximum de Vraisemblance (Maximum Likelihood ou ML en anglais).

$$h^* = \operatorname{ArgMax}_{h \in \mathcal{H}} p_{\mathcal{X}}(\mathbf{x}|h) \quad (2.31)$$

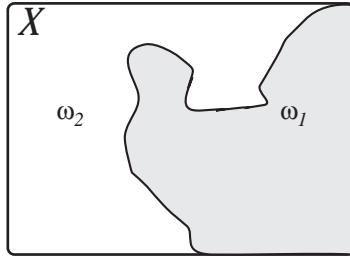


FIG. 2.14 – *Frontière de décision dans l'espace des formes \mathcal{X} .*

Cette règle revient à sélectionner l'hypothèse h pour laquelle l'observation \mathbf{x} est la plus probable, c'est-à-dire l'état du monde qui est le plus à même d'avoir produit l'événement \mathbf{x} . Cela traduit l'idée simple que l'observation \mathbf{x} n'est pas totalement fortuite et était même fortement probable étant donné l'état du monde h .

2.4.3.2 Cas de deux classes : la discrimination

Nous supposons maintenant que la tâche d'apprentissage consiste à discriminer les formes observées \mathbf{x} en deux classes : $\mathcal{H} = \mathcal{F} = \{\omega_1, \omega_2\}$.

Étant donnée l'observation \mathbf{x} , les espérances de risque associées à chaque décision sont respectivement (en notant $l(\omega_i|\omega_j) = l_{ij}$) :

$$\begin{aligned} R(\omega_1) &= l_{11} p(\omega_1|\mathbf{x}) + l_{12} p(\omega_2|\mathbf{x}) \\ R(\omega_2) &= l_{21} p(\omega_1|\mathbf{x}) + l_{22} p(\omega_2|\mathbf{x}) \end{aligned}$$

La règle de décision de Bayes stipule de choisir l'hypothèse d'espérance de risque minimal. Par exemple, il faut décider d'attribuer la forme \mathbf{x} à la classe ω_1 si et seulement si :

$$(l_{21} - l_{11}) p(\omega_1|\mathbf{x}) \geq (l_{12} - l_{22}) p(\omega_2|\mathbf{x}) \quad (2.32)$$

soit encore en utilisant la formule de Bayes de révision des probabilités :

$$(l_{21} - l_{11}) p(\mathbf{x}|\omega_1) p(\omega_1) \geq (l_{12} - l_{22}) p(\mathbf{x}|\omega_2) p(\omega_2)$$

d'où, en passant par le logarithme du rapport :

$$d(\mathbf{x}) = \log \frac{p(\mathbf{x}|\omega_1)}{p(\mathbf{x}|\omega_2)} + \log \frac{(l_{21} - l_{11}) p(\omega_1)}{(l_{12} - l_{22}) p(\omega_2)} \geq 0 \quad (2.33)$$

La règle de décision bayésienne se traduit ainsi par une fonction de décision d (ou fonction de discrimination) décrivant une *frontière* ou *surface de décision* dans l'espace \mathcal{X} , avec d'un côté les formes à attribuer à la classe ω_1 et de l'autre les formes à attribuer à la classe ω_2 (Voir la figure 2.14).

Cette remarque est importante car elle suggère d'*apprendre directement cette fonction de décision*, et la frontière correspondante, plutôt que d'apprendre les probabilités impliquées dans la règle bayésienne de la décision (e.g. la règle (2.30)) qui sont généralement beaucoup plus difficiles à estimer. C'est là la base de toutes les méthodes de classification par détermination de surfaces de décision, dont par exemple une grande partie des méthodes connexionnistes.

On peut noter également que dans le cas particulier de la discrimination entre deux classes de distribution normale de moyennes μ_1 et μ_2 avec des matrices de covariance égales $\Sigma_1 = \Sigma_2 = \Sigma$, la fonction de décision $d(\mathbf{x})$ est une fonction linéaire :

$$d(\mathbf{x}) = (\mathbf{x} - \frac{1}{2(\mu_1 + \mu_2)})^\top \Sigma^{-1} (\mu_1 - \mu_2) + \ln \frac{(l_{21} - l_{11}) p(f_1)}{(l_{12} - l_{22}) p(f_2)} \quad (2.34)$$

2.4.4 Panorama des méthodes inductives dans le cadre bayésien

La théorie de la décision bayésienne fournit une prescription sur l'hypothèse optimale au sens d'un certain risque défini pour un échantillon d'apprentissage donné (à la différence du cadre statistique de Vapnik qui veut se préparer à faire face à tout échantillon possible). Dans ce cadre, le calcul des probabilités *a posteriori* joue un rôle central comme le montrent les équations (2.30) et (2.31) respectivement associées aux règles de décision par Maximum *a posteriori* (minimisant l'espérance de coût) et de Maximum de Vraisemblance (minimisant la probabilité d'erreur). Il s'agit donc de calculer la distribution $p_{\mathcal{F}}(f)$ sur les états du monde – ce qui dans le cas de la classification se traduit par le calcul des probabilités de chaque classe –, et des distributions de probabilité d'appartenance conditionnelle $p_{\mathcal{X}}(\mathbf{x}|f)$. On peut dire qu'une fois que l'on a défini ces règles de décision, tout le reste de la théorie de l'inférence bayésienne est dédié aux méthodes d'estimation de ces probabilités à partir de données d'apprentissage.

Trois familles de méthodes ont été étudiées pour résoudre ce problème.

- Les *méthodes paramétriques* dans lesquelles on suppose *a priori* que les densités de probabilités recherchées ont une certaine forme fonctionnelle. Par exemple, on peut décider qu'il y a de bonnes raisons pour que les densités soient des gaussiennes dont il faut alors identifier la moyenne et la matrice de covariance. L'avantage de ces techniques est qu'elles permettent, grâce à cette connaissance *a priori*, d'obtenir des estimations précises avec peu de données d'apprentissage. L'inconvénient est que la forme fonctionnelle choisie *a priori* peut se révéler inadaptée pour représenter la vraie densité.
- Les *méthodes semi-paramétriques* cherchent à relâcher les contraintes des méthodes paramétriques en s'autorisant une classe générale de formes fonctionnelles pour les densités. L'idée est cependant de pouvoir contrôler cette plus grande flexibilité en réglant des paramètres fonctionnels de manière systématique en fonction des données et de la taille de l'échantillon d'apprentissage. On peut ainsi décider de s'intéresser à des densités représentées par un certain nombre de gaussiennes (*mélanges de distributions*), ou bien choisir de représenter ces densités par des réseaux de neurones dont on contrôle les paramètres comme le nombre de couches ou le nombre de neurones.
- Finalement, les *méthodes nonparamétriques* ne font plus aucune hypothèse sur la forme des distributions recherchées, et permettent à celles-ci d'être entièrement déterminées par les données. Ces méthodes travaillent directement dans l'espace \mathcal{X} des formes par définition de voisinages ou par interpolation. Si on obtient ainsi une flexibilité totale, c'est au prix d'un nombre de paramètres de description qui augmente comme le nombre d'observations et peut rapidement devenir ingérable. Il faut donc prévoir des méthodes de contrôle *a posteriori*. Les techniques par noyaux ou par plus proches voisins sont parmi les plus représentatives de cette famille de méthodes.

Nous détaillons davantage chacune de ces approches au chapitre 17 dans la section 17.2.3 et dans les chapitres concernés, en particulier le chapitre 14.

2.4.5 Et si l'espace des hypothèses ne contient pas la fonction cible?

L'approche bayésienne est souvent décrite comme une approche dans laquelle l'espace des fonctions hypothèse \mathcal{H} , parfois réalisé par une famille de fonctions paramétriques, contient la fonction cible. Ce n'est évidemment pas nécessairement le cas. Que peut-on dire alors si l'on force l'apprentissage dans \mathcal{H} , une famille paramétrée de fonctions ?

On suppose toujours que l'échantillon de données \mathcal{S} est issu d'un tirage i.i.d. (indépendant et identiquement distribué) suivant la densité inconnue $p(\mathbf{x})$, et que l'on cherche à rendre compte

des données par le modèle $p_h(\mathbf{x})$. L'estimation par maximum de vraisemblance fournit alors l'hypothèse $h^* \in \mathcal{H}$ maximisant la fonction de vraisemblance (*log-likelihood function*) :

$$\mathcal{L}_m(h) = \sum_{i=1}^m \log p_h(\mathbf{x}_i) \quad (2.35)$$

Une application de la loi des grands nombres montre que $\mathcal{L}_m(h)/m$ tend, quand m tend vers l'infini, vers l'espérance de $\log p_h(\mathbf{x}_i)$ ($\int_{\mathcal{X}} p \log p_h dx$), avec probabilité 1 (convergence presque sûre).

Pour de nombreuses familles de fonctions, cette expression a un maximum unique h^* dont la densité associée p_{h^*} peut être différente de la vraie valeur p puisque nous avons supposé que p n'appartient pas nécessairement à \mathcal{H} . Il est heureusement possible de dire qu'en un sens, la densité p_{h^*} est la plus « proche » de la vraie valeur p , dans la mesure où elle minimise la *distance de Kullback-Leibler* (encore appelée divergence de Kullback-Leibler) :

$$d_{KL}(p, p_h) = \int_{\mathcal{X}} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{p_h(\mathbf{x})} d\mathbf{x} \quad (2.36)$$

Cette mesure n'est pas symétrique et n'est donc pas une distance au sens mathématique. Il s'agit plutôt d'une dissemblance « dirigée » de la vraie densité à la densité estimée, pour laquelle p_{h^*} est la valeur la « moins fausse » possible. Si la vraie densité appartient à l'espace \mathcal{H} , alors $p(\mathbf{x}) = p_{h^*}(\mathbf{x})$ et $d_{KL}(p, p_{h^*}) = 0$.

2.4.6 En résumé : la procédure inductive bayésienne

Pour résumer, l'approche inductive bayésienne standard sur un échantillon de données est la suivante :

1. Définir l'espace des hypothèses (on parle aussi dans ce cadre de « modèles ») dans lequel on suppose que se trouve la fonction cible.
2. Utiliser la connaissance disponible sur le domaine pour assigner des probabilités *a priori* aux différentes hypothèses possibles et des densités de probabilité sur les paramètres des familles de modèles s'il y a lieu.
3. Utiliser le théorème de Bayes pour calculer les probabilités (ou les densités de probabilité) *a posteriori* étant donné l'échantillon d'apprentissage. Afin de comparer les différents modèles, il est nécessaire de conditionner tous les paramètres intervenant dans les modèles. Ces probabilités conditionnées sont les probabilités des modèles étant donné l'échantillon d'apprentissage, elles fournissent les probabilités relatives des modèles, sans tenir compte de la complexité de chaque modèle.
4. Choisir un algorithme de recherche afin d'explorer efficacement l'espace des modèles possibles pour trouver le modèle de probabilité *a posteriori* maximale (localement).
5. Stopper la recherche lorsque le modèle le plus probable est trouvé, ou lorsqu'il n'est plus rentable de continuer la recherche. Généralement, le critère d'arrêt implique un compromis entre l'optimalité du modèle trouvé et la complexité de la recherche. Il faut le plus souvent avoir recours à des heuristiques pour accélérer la recherche au risque de manquer le modèle optimal.

2.5 Discussion : Quels types d'analyses et de principes inductifs ?

L'induction supervisée consiste à utiliser un échantillon de données pour extrapoler soit la réponse à une nouvelle question (prédiction), soit une règle générale de décision (identification).

Dans cette optique deux questions fondamentales se posent : quel principe inductif faut-il adopter et quelles garanties existe-t-il sur les résultats ?

Parmi les grands principes inductifs immédiatement envisageables, nous avons étudié pour le moment le principe *ERM* dictant de choisir l'hypothèse dont l'adéquation avec les données d'apprentissage est la meilleure (au sens du risque empirique) et le principe bayésien stipulant de choisir l'hypothèse minimisant l'espérance de risque, qui peut aussi, dans le cas de coûts uniformes, être l'hypothèse la plus probable étant donné l'échantillon d'apprentissage.

Il se trouve que ces deux principes inductifs ne conduisent pas à la même analyse. Pour mesurer la pertinence du principe *ERM*, c'est-à-dire la corrélation entre le risque empirique, sur la base duquel est choisie la meilleure hypothèse, et le risque réel, véritable objectif, il faut utiliser un théorème de convergence uniforme faisant intervenir la pire hypothèse possible dans l'espace \mathcal{H} . On obtient alors une analyse dans le pire cas. Cette analyse présente deux aspects positifs. D'une part, elle fournit des garanties extrêmes applicables dans la pire des situations possibles (pire fonction cible, pire choix de la meilleure hypothèse selon *ERM*, pire échantillon d'apprentissage). D'autre part, elle indique que pour établir un lien entre risque empirique et risque réel, il faut tenir compte de la richesse de l'espace des hypothèses. Cela conduit naturellement à envisager des principes inductifs plus puissants que le principe *ERM*, ce sera l'objet de la suite de ce chapitre. La contrepartie de cette analyse dans le pire cas est qu'elle fournit des bornes de différence entre risque empirique et risque réel souvent éloignées de ce qui est observé. On aimerait des analyses plus fines tenant compte de cas typiques ou bien pouvant prendre en compte la distribution des données d'apprentissage. Les approches actuelles sur la classification à large marge par exemple sont un pas dans cette direction (voir le chapitre 9).

Le point de vue bayésien est quant à lui indissociable d'une analyse en cas moyen : cette fois-ci l'hypothèse à retenir est celle qui minimise l'espérance d'erreur en fonction de la probabilité *a priori* des fonctions cible. L'avantage est que la décision résultante est optimale. En revanche, cette analyse presuppose d'une part que l'on sache identifier l'espace dans lequel se trouve la fonction cible, et que, d'autre part, on soit capable de déterminer une distribution de probabilité *a priori* sur cet espace. Si cela est possible, alors le cadre bayésien fournit un moyen intéressant d'expression de la connaissance *a priori* sur le problème. Lorsque l'espace des fonctions hypothèse est mal choisi, alors l'hypothèse prescrite par la décision bayésienne est la plus proche de la fonction cible au sens de la distance de Kullback-Leibler.

Notons avant de passer à des principes inductifs plus fins qu'il existe d'autres types d'analyses. Par exemple, une autre analyse en cas moyen est celle de la physique statistique qui étudie les comportements les plus probables quand on peut voir le problème d'apprentissage comme mettant en jeu un grand nombre d'éléments. D'autres analyses étudient des mesures de performances différentes, par exemple le nombre d'erreurs commises en cours d'apprentissage. Nous renvoyons le lecteur intéressé au chapitre 17 pour plus de détails sur ces approfondissements théoriques.

2.6 Les grands principes inductifs avec régulation des hypothèses

L'examen du compromis biais-variance (section 2.2.1) et l'analyse du principe de minimisation du risque empirique par Vapnik et les théoriciens de l'apprentissage *PAC* ont clairement montré que l'espérance de risque (le risque réel) dépend à la fois du risque empirique mesuré sur l'échantillon d'apprentissage et de la « capacité » de l'espace des fonctions hypothèse. Plus celle-ci est grande, plus il y a de chance de trouver une hypothèse proche de la fonction cible (erreur d'approximation faible), mais plus aussi l'hypothèse minimisant le risque empirique dépend de

l'échantillon d'apprentissage particulier fourni (erreur d'estimation forte), ce qui interdit d'extrapoler avec certitude la performance mesurée par le risque empirique au risque réel.

En d'autres termes, l'induction supervisée doit toujours faire face au risque de « surapprentissage » (*over-fitting*). Si l'espace des hypothèses \mathcal{H} est trop riche, il y a de fortes chances que l'hypothèse retenue, dont le risque empirique est faible, présente un risque réel élevé. Cela est dû au fait que plusieurs hypothèses peuvent avoir un risque empirique faible sur un échantillon d'apprentissage, tout en ayant des risques réels très différents. Il n'est donc pas possible, sur la base du seul risque empirique mesuré, de distinguer les bonnes hypothèses des mauvaises. Il faut donc restreindre autant que possible la richesse de l'espace des hypothèses, tout en cherchant à préserver une capacité d'approximation suffisante.

2.6.1 L'idée générale : le réglage de la classe d'hypothèses

Puisque l'on ne peut mesurer que le risque empirique, l'idée est donc d'essayer d'évaluer le risque réel encouru en corrigeant le risque empirique, nécessairement optimiste, par un *terme de pénalisation* correspondant à une mesure de la capacité de l'espace d'hypothèses \mathcal{H} utilisé. C'est là l'essence de toutes les approches de l'induction révisant le principe de minimisation du risque empirique (l'adaptation aux données) par un terme de *régularisation* (dépendant de la classe d'hypothèses). Cette idée fondamentale se retrouve au cœur de tout un ensemble de méthodes comme la *théorie de la régularisation*, le *principe de longueur de description minimale* (*Minimum Description Length Principle : MDL*), le *critère d'information d'Akaike* (AIC), et d'autres méthodes basées sur des mesures de complexité (par exemple telles que discutées par [Bar91, BC91]).

Le problème ainsi défini est connu, au moins empiriquement, depuis longtemps, et de nombreuses techniques ont été développées pour y faire face. On peut les ranger en trois catégories principales : les méthodes de sélection de modèles, les techniques de régularisation, et les méthodes de moyennage.

- Dans les *méthodes de sélection de modèles*, la démarche consiste à considérer un espace d'hypothèses \mathcal{H} et à le décomposer en une collection discrète de sous-espaces emboîtés $\mathcal{H}_1 \subseteq \mathcal{H}_2 \subseteq \dots \subseteq \mathcal{H}_d \subseteq \dots$, puis, étant donné un échantillon d'apprentissage, à essayer d'identifier le sous-espace optimal dans lequel choisir l'hypothèse finale. Plusieurs méthodes ont été proposées dans ce cadre, que l'on peut regrouper en deux types :
 1. Les *méthodes de pénalisation de la complexité*, parmi lesquelles figurent le *principe de minimisation du risque structurel* (*SRM : Structural Risk Minimization*) de Vapnik [Vap95], le *principe de Longueur de Description Minimale* (*MDL*: *Minimum Description Length principle*) de Rissanen [Ris78] et diverses méthodes ou critères statistiques de sélection [FG94].
 2. Les *méthodes de validation par apprentissages multiples* : validation croisée et boots-trapping.
- Les *méthodes de régularisation* fonctionnent dans le même esprit que les méthodes de sélection de modèles, mis à part qu'elles n'imposent pas une décomposition discrète sur la classe d'hypothèses. À la place, un critère de pénalisation est associé à chaque hypothèse, qui, soit mesure la complexité de leur forme paramétrique, soit mesure des propriétés globales de « régularité », liées par exemple à la dérivabilité des fonctions hypothèse ou à leur dynamique (par exemple des fonctions de haute fréquence, c'est-à-dire changeant de valeur rapidement, seront davantage pénalisées que des fonctions de basse fréquence).
- Les *méthodes de moyennage* ne sélectionnent pas une hypothèse unique dans l'espace \mathcal{H} des hypothèses, mais choisissent une combinaison pondérée d'hypothèses pour former une

fonction de prédiction composée. Une telle combinaison pondérée peut avoir comme effet de « lisser » les hypothèses erratiques (comme dans les méthodes de *moyennage bayésien* et le *bagging*), ou bien d'augmenter le pouvoir de représentation de la classe d'hypothèses si celle-ci n'est pas convexe (comme dans le *boosting*).

Toutes ces méthodes ont généralement conduit à des améliorations notables de performances par rapport à des méthodes « naïves ». Cependant, elles demandent d'être utilisées avec soin. D'une part, en effet, elles correspondent parfois à une augmentation de la richesse de l'espace d'hypothèses, donc à des risques accrus de surapprentissage. D'autre part, elles requièrent souvent de l'expertise pour être appliquées, en particulier parce qu'il faut régler des paramètres supplémentaires. Certains travaux récents essaient pour ces raisons de déterminer automatiquement la complexité appropriée des hypothèses candidates pour s'adapter aux données d'apprentissage.

2.6.2 La sélection de modèles

Nous allons définir plus formellement le problème de la sélection de modèles qui est l'objectif de toutes ces méthodes.

Soit une séquence enchâssée d'espaces ou classes d'hypothèses (ou *modèles*) $\mathcal{H}_1 \subseteq \mathcal{H}_2 \subseteq \dots \subseteq \mathcal{H}_d \subseteq \dots$ où les espaces \mathcal{H}_d sont de capacité croissante. La fonction cible f peut ou non être inclue dans l'une de ces classes. Soit h_d^* l'hypothèse optimale dans la classe d'hypothèses \mathcal{H}_d , et $R(d) = R_{\text{Réel}}(h_d^*)$ le risque réel associé. Nous noterons que la séquence $R(d)_{1 \leq d \leq \infty}$ est décroissante au sens large puisque les classes d'hypothèses \mathcal{H}_d sont emboîtées, et donc que leur capacité d'approximation de la fonction cible f ne peut que s'améliorer.

À l'aide de ces notations, le problème de la sélection de modèles peut se définir comme suit.

Définition 2.8 (Le problème de la sélection de modèle)

Le problème de sélection de modèle consiste à choisir, sur la base d'un échantillon d'apprentissage \mathcal{S} de longueur m , une classe d'hypothèses \mathcal{H}_{d^*} et une hypothèse $h_d \in \mathcal{H}_{d^*}$ telles que le risque réel associé $R_{\text{Réel}}(h_d)$ soit minimal.

La conjecture sous-jacente est que le risque réel associé aux hypothèses retenues h_d pour chaque classe \mathcal{H}_d présente un minimum global pour une valeur non triviale de d (c'est-à-dire différente de zéro et de m) correspondant à l'espace d'hypothèses \mathcal{H}_{d^*} « idéal ». (Voir figure 2.15).

Il s'agit donc d'une part de trouver l'espace d'hypothèses \mathcal{H}_{d^*} idéal, et d'autre part de sélectionner la meilleure hypothèse h_d à l'intérieur de \mathcal{H}_{d^*} . La définition ne se prononce pas sur ce dernier problème. Il est généralement résolu en utilisant le principe *ERM* dictant de rechercher l'hypothèse de risque empirique minimal.

Pour la sélection de \mathcal{H}_{d^*} , on utilise une estimation du risque réel optimal dans chaque \mathcal{H}_d en sélectionnant la meilleure hypothèse selon le risque empirique (méthode *ERM*) et en corrigeant le risque empirique associé par le terme de pénalisation lié aux caractéristiques de l'espace \mathcal{H}_d . Le problème de sélection de modèle revient donc à résoudre une équation du type:

$$\begin{aligned} d^* &= \underset{d}{\text{ArgMin}} \quad \{ h_d \in \mathcal{H}_d : R_{\text{Réel}}^{\text{Estimé}}(h_d) \} \\ &= \underset{d}{\text{ArgMin}} \quad \{ h_d \in \mathcal{H}_d : R_{\text{Emp}}(h_d) + \text{terme-de-pénalisation} \} \end{aligned} \tag{2.37}$$

L'idée générale pour implémenter ces méthodes de pénalisation est d'avoir un **algorithme d'apprentissage** retournant une hypothèse h_d candidate pour chaque classe d'hypothèses \mathcal{H}_d (par

exemple en suivant le principe *ERM*), puis d'avoir un **algorithme de sélection de modèle** choisissant le meilleur espace d'hypothèses associé (suivant l'équation 2.37). L'hypothèse finale est alors l'hypothèse candidate dans cet espace.

Notons que le choix du meilleur espace d'hypothèses dépend de la taille m de l'échantillon de données. Plus celle-ci est grande, plus il est possible, si nécessaire, de choisir sans risque (c'est-à-dire avec une variance ou un intervalle de confiance faible) un espace d'hypothèses riche permettant d'approcher au mieux la fonction cible f .

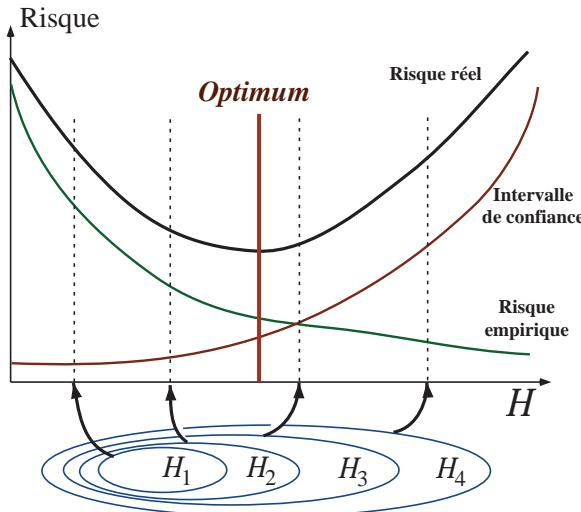


FIG. 2.15 – La borne sur le risque réel résulte de la somme du risque empirique et d'un intervalle de confiance dépendant de la capacité de l'espace d'hypothèses associé. En supposant que l'on dispose d'une séquence enchaînée d'espaces d'hypothèses indicés par d et de capacité croissante, le risque empirique optimal accessible diminue avec les d croissants (le biais), tandis que l'intervalle de confiance (correspondant à la variance) diminue. La borne minimale sur le risque réel est atteinte pour un espace d'hypothèses approprié \mathcal{H}_d .

2.7 Discussion et perspectives

Ce chapitre a introduit l'analyse des facteurs entrant en jeu dans l'induction, et l'étude de divers principes inductifs raisonnables. À l'examen, ceux-ci transforment un problème d'apprentissage en un problème d'optimisation en fournissant un critère que doit optimiser l'hypothèse idéale. La plupart des méthodes d'apprentissage peuvent alors être vues comme des manières de spécifier l'espace des hypothèses à considérer ainsi que la technique d'exploration de cet espace en vue d'y trouver la meilleure hypothèse. Cette vision de l'apprentissage est d'une grande force. Elle permet de concevoir des méthodes d'apprentissage, de les comparer entre elles, et même de construire de nouveaux principes inductifs, comme ceux qui contrôlent automatiquement l'espace d'hypothèses. Il est facile de se laisser séduire et de se mettre à raisonner dans les termes de cette approche. Pourtant, en y réfléchissant, il s'agit là d'un cadre surprenant pour aborder l'apprentissage.

D'une part, il y a une Nature indifférente qui distille des messages, les données, de manière aléatoire, excluant par là les situations d'apprentissage organisées ou du moins bienveillantes. D'autre part, il y a un apprenant solitaire, complètement passif, qui attend les messages, et, en

général, ne fait rien avant de les avoir tous collectés. On évacue ainsi les apprentissages continus, collaboratifs, avec une évolution de l'apprenant. De même sont exclus les apprentissages dans des environnements non stationnaires, un comble pour une science qui devrait avant tout être une science de la dynamique. De plus, l'apprenant cherche à être performant en moyenne (il optimise une espérance de risque), mais il ne cherche pas vraiment à identifier le concept cible. Car dans ce cas, il aurait sans doute intérêt à consacrer ses ressources aux régions de l'espace dans lesquelles la fonction cible présente une grande dynamique (de fortes variations) et moins là où les choses se passent tranquillement. Cela reviendrait à avoir un espace d'hypothèses à géométrie variable: riche dans les régions de forte dynamique et pauvre ailleurs. Par ailleurs, le rôle des connaissances *a priori*, si important dans les apprentissages naturels, est ici réduit à une expression très pauvre, puisqu'elle ne concerne que le choix de l'espace d'hypothèses. Finalement, les critères de performances ne prennent en compte que l'espérance d'erreur ou de risque, et pas du tout des critères d'intelligibilité ou de fécondité des connaissances produites. À la réflexion, donc, on est loin d'un cadre d'analyse rendant compte de toute la diversité des situations d'apprentissage. Pour autant, ce cadre très épuré se révèle d'une grande efficacité dans l'analyse de données, ce qui correspond à un vaste champ d'applications.

Ce chapitre a présenté les grandes lignes de l'analyse théorique de l'induction qui permettent d'aborder la suite de l'ouvrage. Le chapitre 17 fournit des développements de cette théorie utiles pour comprendre les recherches actuelles sur l'apprentissage. Ils concernent en particulier :

- Une généralisation de l'analyse du principe *ERM* à des espaces d'hypothèses et à des fonctions de perte quelconques. Il s'agit de l'analyse de Vapnik, si influente sur les travaux récents.
- Une description du principe inductif par compression d'informations non décrit dans le présent chapitre. Ce principe très original prend en compte la quantité d'informations dans les données et les hypothèses, ce qui semble naturel à la réflexion. En revanche il ne s'appuie pas sur la distribution des exemples comme le principe *ERM* ou l'analyse bayésienne.
- Une introduction à l'analyse de l'apprentissage « en-ligne » (*on-line learning*) dans lequel l'apprenant doit réagir après chaque nouvel exemple. Cette approche est intéressante parce qu'elle introduit de nouveaux critères de performance sur l'apprentissage et permet d'envisager des apprenants ayant une certaine initiative dans le choix des exemples fournis.

Le chapitre 17 se terminera, et terminera cet ouvrage, de manière appropriée en discutant la valeur relative des méthodes inductives. Peut-on dire qu'il y en a de meilleures que d'autres ? D'où provient le pouvoir inductif ? Nous retrouverons là certaines des interrogations récurrentes des philosophes de la connaissance.

2.8 Notes historiques et bibliographiques

Dire que l'apprentissage inductif est un problème d'optimisation qui conjugue un principe ou critère inductif à satisfaire au mieux et une méthode de recherche dans un espace d'hypothèse est presque devenu un dogme. Faire ressortir qu'il existe essentiellement trois types de principes inductifs : la minimisation du risque empirique (*ERM*), la théorie bayésienne de la décision qui se traduit souvent par un principe de maximum de vraisemblance, et le principe de compression maximal de l'information, ne suscite pas non plus de surprise. Pourtant, il a fallu beaucoup de temps pour que cette vision de l'apprentissage s'impose, comme pour qu'il soit admis que ces principes, en particulier la minimisation du risque empirique, devaient être examinés pour voir s'ils conduisaient bien à la meilleure induction possible.

La théorie bayésienne de l'apprentissage s'est développée presque naturellement durant le XX^e siècle et en particulier depuis les années 1960. Elle conduit à la notion d'erreur bayésienne optimale, mais elle requiert une connaissance du modèle statistique sous-jacent. Nous recommandons à ce sujet la lecture des ouvrages de référence [Bis95, DHS01, Rip96, Web99].

L'analyse de la pertinence du principe *ERM* a été plus longue à être perçue comme nécessaire et la démarche a été plus accidentée. Il a d'abord fallu que l'idée qu'il était intéressant détudier directement la convergence du risque empirique se fasse jour. Dans le cas des algorithmes réalisant l'*ERM*, cela a débouché sur les cadres *PAC* (Probablement Approximativement Correct) et *VC* (Vapnik-Chervonenkis).

Le cadre *PAC* a été introduit par le papier très influent de Valiant en 1984 [Val84a] dans lequel il étudiait ce qui se révélerait un cas particulier de la convergence du risque empirique où l'espace d'hypothèses est celui de formules logiques et est supposé contenir le concept cible. Cela simplifiait considérablement les choses car, d'une part, le nombre d'hypothèses restait fini même s'il pouvait croître exponentiellement avec le nombre d'attributs et, d'autre part, on pouvait n'examiner que les hypothèses de risque empirique nul. Ce cadre incluait aussi un critère de complexité calculatoire sur l'apprentissage, imposant que la complexité reste polynomiale en un certain nombre de paramètres. Cependant, cet aspect du modèle *PAC* qui a permis de démontrer de nombreux théorèmes de non apprenabilité (en les ramenant à des problèmes de cryptographie) est pratiquement tombé en désuétude. Par ailleurs, afin de s'affranchir de la contrainte que le concept cible doive appartenir à l'espace d'hypothèses, un cadre généralisé a été proposé, appelé apprentissage agnostique. On n'en parle plus car il a été généralisé par l'approche de Vapnik.

En effet, pendant ce temps, en URSS, Vapnik et Chervonenkis, sous l'influence de Kolmogorov, étudiaient depuis les années 1960 le problème général de la convergence des moyennes empiriques vers leur espérance. Ils prouvèrent ainsi que la convergence des espérances de risque est équivalente à la convergence uniforme des fréquences vers des probabilités sur un domaine fini d'événements. C'est ce qui est appelé le théorème clé de la théorie statistique de l'apprentissage. Les premières bornes sur le risque réel en fonction du risque empirique furent prouvées pour la première fois par Vapnik et Chervonenkis en 1974. L'analyse montra que la convergence du risque empirique vers le risque réel fait intervenir une fonction de croissance de l'espace d'hypothèses. Comme cette fonction est très difficile à calculer, il est pratique de la caractériser par un nombre : la dimension de Vapnik-Chervonenkis. Les premiers travaux introduisant cette mesure sont ceux de Vapnik et Chervonenkis en 1971, et, indépendamment, de Sauer (1972) et de Shela (1972). L'introduction de la théorie de Vapnik et Chervonenkis s'est faite grâce à un papier exceptionnel du « four german gang »¹¹ [BEHW89] qui a eu un grand impact dans la communauté de la théorie de l'apprentissage (COLT : *Computational Learning Theory*).

L'analyse de Vapnik, largement popularisée par son livre de 1995 ([Vap95]), a fait prendre conscience à la communauté de l'apprentissage artificiel de l'importance cruciale de la définition et de la caractérisation de l'espace d'hypothèses. Depuis longtemps les praticiens savaient en effet qu'il leur fallait contrôler la complexité de leur modèle d'apprentissage pour ne pas tomber victime de surapprentissage, c'est-à-dire d'apprentissage par cœur sans généralisation. Depuis 1982, ils avaient admis, sous l'influence du papier de Mitchell ([Mit82]), qu'il fallait que l'espace d'hypothèses soit contraint par un biais. Cependant, c'est vraiment l'analyse de Vapnik qui a fourni un cadre conceptuel complet permettant de comprendre au moins heuristiquement le compromis entre risque empirique et capacité de l'espace d'hypothèses. Il faut cependant noter l'influence des papiers sur le compromis biais-variance ([GBD92]).

Pour toutes ces questions, nous reportons le lecteur aux ouvrages [CM98, Hay99, Vap95,

11. Selon l'expression de Manfred Warmuth, l'un des quatre auteurs, et un théoricien éminent et inventif.

KV94]. D'autres ouvrages sont plus techniques mais sont essentiels pour ceux qui veulent aller plus loin dans cette étude: [AB92, AB96, DGL96, Vid97]. Un ouvrage très intéressant sur des points de vue multiples de la théorie de l'apprentissage est [Wol95].

Résumé

Ce chapitre a montré que l'induction peut-être formalisée par un jeu entre une Nature produisant des exemples étiquetés selon une fonction cible, et un apprenant cherchant à approcher cette fonction cible par une fonction hypothèse de manière à minimiser l'espérance de risque appelée risque réel. Pour ce faire, l'apprenant utilise un principe inductif lui dictant quelle hypothèse il doit choisir étant donnés les exemples d'apprentissage, et un algorithme de recherche effectif dans l'espace d'hypothèses. Ce chapitre s'est penché sur l'analyse des principes inductifs.

Il existe trois grands principes inductifs de base: le principe de minimisation du risque empirique qui dicte de choisir l'hypothèse qui minimise le risque sur l'échantillon d'apprentissage; le principe bayésien qui stipule de choisir l'hypothèse minimisant l'espérance de risque, ce qui revient souvent à prendre l'hypothèse dont la vraisemblance est maximale étant donnés les exemples; finalement le principe de compression d'information qui prescrit de choisir l'hypothèse permettant de transmettre l'information contenue dans les exemples d'apprentissage de la manière la plus économique. Les deux premiers ont été décrits et analysés plus en détail dans ce chapitre. En particulier, le principe *ERM* se prête naturellement à une analyse dans le pire cas, tandis que le principe bayésien, prenant en compte la distribution *a priori* des fonctions cible, conduit à une analyse en moyenne.

L'une des conséquences les plus remarquables de ces analyses est qu'elles soulignent l'importance cruciale de l'espace d'hypothèses considéré dans la confiance que l'on peut accorder aux inductions réalisées. Il faut un espace d'hypothèses suffisamment riche pour pouvoir approcher la fonction cible d'assez près, mais il ne faut pas qu'il le soit trop sous peine de conduire à des hypothèses apparemment bonnes sur les données d'apprentissage, mais mauvaises en réalité. La mise en évidence de ce compromis a amené à reconsidérer les principes inductifs pour en faire des principes inductifs avec contrôle et ajustement automatique de l'espace d'hypothèses.

Chapitre 3

L'environnement méthodologique de l'apprentissage

D'un point de vue conceptuel, l'apprentissage se joue entre un espace de description des objets d'entrée et un espace d'hypothèses. Le choix d'un principe inductif permet d'évaluer, à partir des exemples, la qualité des hypothèses et de prescrire l'hypothèse théorique optimale. Pour qu'une méthode d'apprentissage soit effective, il faut spécifier un algorithme de recherche dans l'espace des hypothèses qui tentera d'identifier une hypothèse optimale ou du moins de s'en approcher.

Ce chapitre a pour objectif de fournir les bases permettant la réalisation d'une méthode d'apprentissage en partant du problème du choix de la représentation des entrées et de celle des hypothèses, puis en dressant un panorama des techniques de recherche et d'optimisation utilisables dans le contexte de l'apprentissage, enfin en examinant les méthodes d'évaluation et de validation des résultats obtenus à l'issue de l'apprentissage. Le problème du test et de la comparaison empirique des algorithmes est également discuté.

Ce chapitre se clôt en dressant une typologie des méthodes adaptées à chaque classe de problèmes.

EST-IL FACILE de définir un cygne ou une oie ? On pourrait penser que oui. Les naturalistes ont accumulé des connaissances sur cette question et les ont largement vulgarisées. Essayons donc avec notre dictionnaire usuel. Voici par exemple ce qui se trouve dans le *Petit Larousse*, édition 2000.

Cygne : oiseau palmipède *ansériforme* au long cou souple, migrateur.

Oie : oiseau palmipède massif au long cou et au bec large.

Ansériforme : oiseau, généralement palmipède, à l'allure de *canard*, mais dont certaines espèces sont des échassiers à bec crochu, tel que le *kamichi* et les *anatidés*. Les ansériformes forment un ordre.

Anatidé : oiseau palmipède au corps massif et au bec aplati, tel que le *canard*, l'*oie*, le *cygne*. Les anatidés forment une famille de l'ordre des ansériformes.

Canard : oiseau palmipède de la famille des anatidés, bon voilier et migrateur à l'état sauvage. Le canard cancane.

Kamichi : oiseau échassier des marais et des prairies humides de Patagonie, aux ailes armées de deux éperons. Longueur: 90 cm, genre *Chauna*, ordre des ansériformes, famille des anhimidés.

Anhimidé ... (n'est pas une entrée dans ce dictionnaire.)

Bon... à moins que l'on soit dans un marais de Patagonie face à un échassier aux ailes armées, tout cela n'est pas très utile pour identifier un oiseau. Ces définitions circulaires masquent les données et les concepts sous des niveaux d'abstraction tellement différents qu'en pratique elles sont inopérantes. De plus, la variété des contextes (des biais d'apprentissage) est également importante: il faut au lecteur de grandes connaissances *a priori*, et très bien organisées. Par exemple la notion de *migrateur* est importante dans ces définitions et elle est supposée connue, alors que ce n'est pas une évidence à l'observation d'un oiseau... et à quoi peut bien servir de connaître le mot désignant le cri du canard pour caractériser cet animal ?

Alors, comment écrire un programme qui saurait apprendre à distinguer un cygne d'une oie ? La réponse est qu'il faudra être plus modeste, c'est-à-dire soigneusement délimiter un cadre opérationnel par la définition de biais d'apprentissage. Rappelons l'exemple de l'avant-propos : l'univers est réduit à un lac sur lequel on impose que seulement deux espèces d'oiseaux puissent nager. Les observations aussi sont limitées à la taille et à la couleur. On ne cherche pas à définir la nature du cygne ou de l'oie de manière universelle: on n'a pour ambition que d'apprendre à les distinguer sous des conditions fixées de manière stricte.

Prenons maintenant sur un oiseau l'ensemble des attributs suivants :

- la taille ;
- le fait qu'il vole ou non ;
- la couleur de son bec ;
- son chant ;
- son *genre*.¹

Ainsi la liste (152 cm, vole, « couac », bec jaune, genre *Anatidae*) nous indiquera, l'hiver dans nos régions, un cygne chanteur (*Cygnus Cygnus L.*) et la liste (110 cm, ne vole pas, « krrr », bec noir, genre *Aptedonytes*) se rapporte plutôt à un manchot, mais n'est pas assez complète pour que l'on sache de quelle espèce il s'agit.

Une autre question : est-il facile de définir une carte à jouer ? Mais oui. Il suffit de noter sa couleur et son rang, qui peuvent prendre respectivement leur valeurs dans les domaines {♠, ♠, ♦, ♣}

1. Dans la hiérarchie de Linné, cette variable est au dessus de l'*espèce* et au dessous de la *famille*, elle-même au dessous de l'*ordre*.

et $\{A, R, D, V, 10, 9, 8, 7, 6, 5, 4, 3, 2\}$. Cette définition est parfaite, puisque les cartes à jouer sont des objets dont le sens est par nature complètement décrit par ces deux caractéristiques. En revanche, aucun oiseau ne porte le nom de son espèce sur son plumage.

C'est que les noms donnés aux formes de la nature sont symboliques : ils sont une abstraction qui regroupe des individus selon des contrastes avec d'autres individus. Ces concepts ont été extraits d'une multitude de descripteurs souvent numériques, comme la taille, ou un peu plus abstraits comme la couleur, ou très élaborés, comme le fait d'être *migrateur* ou non. Autrement dit, les connaissances sur les individus doivent être symbolisées si l'on veut en tirer profit pour en extraire une définition opératoire.

Cette introduction n'a pour but que de rappeler que l'apprentissage artificiel doit évidemment se poser le problème de la symbolisation ou de la représentation des connaissances, qui est comme on le sait une des questions centrales de l'intelligence artificielle. Cette question est plus aisée à résoudre pour des données « artificielles » comme les cartes à jouer que pour des données naturelles, évidemment plus intéressantes.

Le plan de ce chapitre

Ce chapitre est centré sur la nature de la représentation des connaissances pour l'apprentissage et sur la façon d'utiliser ces représentations.

Il y a en pratique deux problèmes qui se posent d'entrée :

- Comment représenter les objets ?
- Comment représenter les hypothèses faites par le programme d'apprentissage ?

Ils seront traités dans les deux premières parties de ce chapitre.

Mais les aspects méthodologiques généraux de l'apprentissage ne se limitent pas à ces deux problèmes. Une fois choisies la manière de représenter les objets et celle de formuler les hypothèses, il se pose la question suivante :

Étant donné l'ensemble des hypothèses et un échantillon d'apprentissage, comment trouver la meilleure hypothèse ?

Nous avons vu au chapitre 2 que la notion de qualité d'une hypothèse pouvait être abordée de diverses manières. Nous étudierons dans la troisième partie par quelles techniques d'optimisation on peut rechercher la meilleure, sans préjuger de la mesure de qualité employée.

Le dernier problème général est celui de l'évaluation de l'hypothèse trouvée. Même en supposant l'étape précédente parfaitement réalisée, comment estimer la qualité véritable de cette hypothèse ? Cette question sera traitée en quatrième partie de ce chapitre. En particulier, nous verrons au paragraphe 3.4.4 une illustration du fait présenté au chapitre précédent : si le critère choisi est la minimisation du risque empirique (*ERM*), la complexité de la classe d'hypothèses choisie est un paramètre important à maîtriser pour éviter le sur-apprentissage.

Deux autres aspects seront abordés dans ce chapitre : celui du prétraitement des données d'apprentissage, qui a pour objet de mettre les algorithmes dans des conditions les meilleures possibles et celui de la sélection des attributs qui traite de la réduction de l'espace de représentation des objets.

Notations utiles pour le chapitre

$P(X = VRAI)$ ou $P(X)$	La probabilité que l'événement X soit <i>VRAI</i>
\mathcal{X}	L'espace de représentation des objets (des formes)
\mathcal{U}	L'espace de supervision (des sorties désirées)
\mathcal{S}	L'échantillon d'apprentissage (ensemble ou séquence d'exemples)
\mathcal{T}	L'échantillon de test
\mathcal{V}	L'échantillon de validation
m	La taille d'un échantillon d'apprentissage (le nombre d'exemples)
$\mathbf{z}_i = (\mathbf{x}_i, \mathbf{u}_i)$	Un exemple (élément d'un échantillon d'apprentissage)
\mathbf{x}_i	La description d'un objet dans un espace de représentation
\mathbf{u}_i	La supervision, ou sortie désirée, d'un exemple
$f : \mathcal{X} \rightarrow \mathcal{U}$	La fonction cible (celle que l'on cherche à apprendre)
\mathcal{C}	L'ensemble des classes
C	Le nombre de classes
ω_i	Une classe de \mathcal{C}
Υ_{ij}	La surface séparatrice entre les classes ω_i et ω_j
\mathcal{H}	L'espace des hypothèses d'apprentissage
$h \in \mathcal{H}$	Une hypothèse produite par un apprenant
$l(f, h)$	La perte (distance) entre la fonction cible et une hypothèse
$R_{Emp}(h)$	Le risque empirique d'une hypothèse
$R_{Réel}(h)$	Le risque réel d'une hypothèse

3.1 L'espace des données d'apprentissage

L'apprentissage s'appuie sur des données (des objets) qu'il faut représenter. Suivant le type de ces données, certaines représentations sont plus ou moins adaptées. Par ailleurs, toute description des données suppose déjà un prétraitement, ne serait-ce que dans le choix des attributs de description ou la manière de faire face à des données imparfaites.

3.1.1 La représentation des objets de l'apprentissage

Les connaissances sur les données elles-mêmes sont symbolisées grâce à un espace de représentation des données noté \mathcal{X} . C'est dans cet espace que s'effectue la description des objets. Dans l'exemple d'introduction de ce livre, les oies et les cygnes sont représentés par deux chiffres : leur taille et leur niveau de gris. Par conséquent chaque objet (chaque oiseau) est représenté par deux valeurs numériques, ou par un point du plan, ou encore par un vecteur de \mathbb{R}^2 . Dans l'exemple en tête de ce chapitre, un oiseau est représenté différemment : par cinq attributs de natures diverses.

Le premier exemple est très fréquent : la description d'un objet par d valeurs numériques, donc l'utilisation de $\mathcal{X} = \mathbb{R}^d$ comme espace de représentation, permet en effet d'utiliser des outils analytiques, géométriques, probabilistes, etc.

Il y a un autre cas courant : celui où les données sont représentées par un vecteur binaire. Ceci correspond au cas où l'on décrit les objets à l'aide d'une série de tests et où chaque objet est *VRAI* ou *FAUX* vis-à-vis de chaque test. Si l'on veut traiter des problèmes complexes, il faut naturellement un grand nombre de descripteurs binaires, parfois des centaines ou des milliers. Les propriétés de cet espace sont formalisées par la logique booléenne ou logique des propositions. La structure algébrique de ces espaces est forte, mais les notions de continuité et de densité de probabilité sont non définies.

Définition 3.1 (Espace de représentation)

L'espace de représentation est noté \mathcal{X} et ses éléments sont appelés données, instances ou objets². Un exemple $\mathbf{z}_i = (\mathbf{x}_i, \mathbf{u}_i)$ est un objet associé à sa supervision.

Les éléments de \mathcal{X} peuvent souvent être détaillés comme un ensemble de d attributs ou descripteurs³ :

$$\mathbf{x} = \{\mathbf{x}_1, \dots, \mathbf{x}_i, \dots, \mathbf{x}_d\}$$

Nous emploierons aussi la notion de distance sur l'ensemble des valeurs que peut prendre un attribut. Rappelons la définition de ce terme :

Définition 3.2 (Distance)

Une distance Δ sur un espace $E \times E$ est une application de $E \times E$ dans \mathbb{R}^+ si et seulement si elle vérifie les propriétés :

- $\Delta(x, y) = 0 \iff x = y$
- $\forall x, y \in \Sigma, \Delta(x, y) = \Delta(y, x)$ (*symétrie*)
- $\forall x, y, z \in \Sigma \quad \Delta(x, y) \leq \Delta(x, z) + \Delta(z, y)$ (*inégalité triangulaire*)

L'inégalité triangulaire n'est pas toujours facile à définir dans les applications pratiques. Une application de $E \times E$ dans \mathbb{R}^+ qui vérifie au plus les deux premiers axiomes est parfois appelée *dissemblance*. Par abus de langage, le mot *distance* est souvent employé indifféremment pour ces deux concepts, en particulier en apprentissage. Nous serons par la suite aussi rigoureux que possible dans l'utilisation de ces deux termes.

Nous allons maintenant passer en revue les types d'attributs auxquels nous aurons affaire par la suite, les cas binaires et numériques étant les plus naturels et les plus simples.

3.1.1.1 La nature des attributs

Nous nous intéressons dans la suite de cet ouvrage aux attributs des types suivants :

Binaire

L'objet \mathbf{x} est décrit par d attributs \mathbf{x}_i dont chacun vaut 1 ou 0, autrement dit *VRAI* ou *FAUX*.

$$\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_i, \dots, \mathbf{x}_d\} = \{0, 1\}^d = \mathbb{B}^d$$

Dans le cas où les d attributs de \mathcal{X} sont tous binaires, les données peuvent être représentées par une matrice binaire ($m \times d$). Cette représentation a des interprétations mathématiques

2. Le terme « données » est vague, mais fait le lien avec l'apprentissage pour la fouille de données. Le terme « instance » est un anglicisme imprécis (souvent un objet, parfois un exemple).

3. En reconnaissance des formes, le terme « paramètre » est parfois employé, comme mauvaise traduction de *feature*.

diverses : logique, algébrique (construction d'un treillis de Galois : chapitre 4, paragraphe 4.5), topologique (notion de distance), informatique (bases de données, voir chapitre 15), etc.

Par exemple, pour quelques espèces d'animaux :

	<i>vole</i>	<i>a des plumes</i>	<i>pond des œufs</i>
oie	1	1	1
ornithorynque	0	0	1
rhinolophe	1	0	0
cygne	1	1	1

Nominal (ou catégoriel)

Par définition, un attribut de ce type appartient à un ensemble fini et non ordonné⁴. Par exemple la « couleur » {♠, ♥, ♦, ♣} d'une carte à jouer est un attribut nominal dans la plupart des cas : d'une part elle ne peut prendre que quatre valeurs et d'autre part il n'y a pas d'ordre sur les couleurs. De même, une pièce au jeu d'échecs peut être de six formes différentes, mais, *grosso modo*, chacune peut s'emparer de chaque autre : elles n'ont pas d'ordre naturel de ce point de vue.

Dans certains cas, une distance ou une dissemblance peut se définir sur l'ensemble des valeurs que peut prendre un attribut nominal. Par exemple, l'ensemble des sons (ou *phonèmes*) de la langue française est un ensemble nominal : il n'est pas ordonné, mais on sait par exemple que le son /a/ est plus proche du son /in/ que du son /k/. Dans cet exemple, la propriété de l'inégalité triangulaire n'est pas vérifiée.

Nominal arborescent

Il existe parfois une hiérarchie naturelle, mais pas un ordre total, sur les valeurs que peuvent prendre un attribut nominal. Par exemple, les groupes sanguins et facteurs rhésus sont au nombre de huit :

$$\{O+, O-, A+, A-, B+, B-, AB+, AB-\}$$

Du point de vue de la compatibilité pour la transfusion, $O+$ est « supérieur » à $A+$, $B+$ et $AB+$, puisque du sang $O+$ peut être transfusé aux trois autres groupes et pas l'inverse. En revanche, du point de vue de cette relation d'ordre, on ne peut rien dire sur le couple ($A+$, $B+$) ni sur le couple ($A+$, $A-$).

Un autre exemple est celui de la couleur, donné en figure 3.1. Nous l'utiliserons au chapitre 4.

Nominal totalement ordonné

Il est en réalité souvent possible de trouver une relation d'ordre sur un attribut nominal. La question est de savoir si elle est utile au problème ou non. Par exemple, si on s'intéresse à l'attribut **couleur** dans un catalogue de voitures, une relation d'ordre semble difficile à définir (le *bleu sprint* est-il supérieur ou inférieur à l'*orange calypso*?). En revanche, en astrophysique, la couleur est caractérisée par une longueur d'onde dans un certain intervalle : c'est un attribut numérique totalement ordonné, donc un intervalle de \mathbb{R} .

De même, dans certains jeux de cartes, les couleurs sont rangées dans un ordre décroissant : le ♠ l'emporte sur le ♥ qui l'emporte sur le ♦ qui l'emporte enfin sur le ♣.

Un attribut nominal totalement ordonné est assimilable à un intervalle de \mathbb{R} ou de \mathbb{N} et peut donc être muni d'une distance.

4. Un attribut est *ordinal* quand il appartient à un ensemble ordonné, mais sur lequel on ne peut pas définir une distance, comme *médaille* ∈ {or, argent, bronze}. La confusion entre les termes « nominal » et « ordinal » est fréquente.

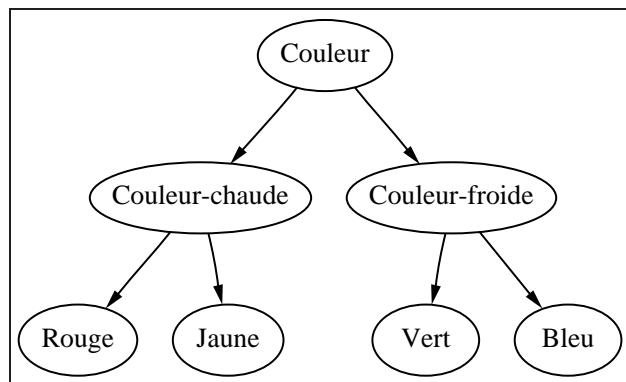


FIG. 3.1 – Une description arborescente possible pour l'attribut Couleur.

Séquenciel nominal

Un texte français est une séquence composée à partir d'un ensemble (un alphabet) d'une centaine de caractères : les cinquante-deux lettres minuscules et majuscules, l'intervalle (le blanc), quelques lettres accentuées, les signes de ponctuation, parfois des abréviations comme :- ou €, etc. Évidemment, l'ordre de ces éléments nominaux est essentiel : la séquence « Le commandant Cousteau. » et la séquence « Tout commença dans l'eau. » sont différentes, bien que composées exactement des mêmes lettres⁵.

On sait munir l'ensemble des valeurs que peut prendre un tel attribut d'une distance, en particulier quand l'ensemble des éléments qui composent la séquence (l'alphabet) est lui-même muni d'une distance.

Séquenciel numérique

La cote boursière de tel ou tel titre est un exemple d'attribut séquenciel numérique : à chaque instant de temps significatif, une valeur numérique est donnée. On peut ainsi produire des séquences de plusieurs centaines de chiffres représentant l'évolution d'un cours sur une année.

Le cas de vecteurs d'attributs arrivant en séquence est typique des problèmes de traitement du signal, comme la parole : chaque centième de seconde est caractérisé après analyse spectrale par un élément de \mathbb{R}^d , d valant typiquement entre 10 et 20.

3.1.1.2 Représentations homogènes et représentations mixtes

L'espace de représentation \mathcal{X} est souvent composé de d attributs de la même nature, généralement dans ce cas binaires ou numériques. Il existe aussi des espaces de représentation composés de plusieurs attributs séquencIELS nominaux : par exemple dans les problèmes d'apprentissage de traducteurs, où l'on doit disposer de couples de phrases.

Dans les cas précédents, \mathcal{X} est homogène : ses d attributs sont tous de même nature. Beaucoup de méthodes d'apprentissage ne peuvent s'appliquer que sur des données décrites dans un espace de représentation homogène.

Mais le cas le plus général est celui où l'espace de représentation $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_i, \dots, \mathbf{x}_d\}$ est mixte, autrement dit composé d'attributs de natures différentes. C'est le cas de la description d'un oiseau donnée ci-dessus pour un cygne chanteur : (152 cm, vole, « couac », bec

5. Dans ce cas précis, les espaces ne sont pas comptés, les accents ou cédilles non plus et les caractères minuscules et majuscules ne sont pas distingués.

jaune, genre *Anatidae*). Le premier attribut est numérique, le second est binaire, le troisième séquentiel et le dernier hiérarchique.

De même le diagnostic sur un patient entrant dans un hôpital porte sur une représentation non homogène de son état. Il pourra être décrit par exemple par les attributs suivants :

- Vacciné contre la diphtérie? Et si oui, il y a combien de temps?
- Température?
- Groupe sanguin?
- Description du type d'affection cutanée?
- Région et type de douleur?
- ...

Peu de méthodes d'apprentissage sont capables d'extraire un concept obtenu par un apprentissage coordonné sur des attributs de natures diverses. La plupart du temps, un concept appris à partir d'exemples mixtes est une combinaison booléenne de propriétés binaires extraites des attributs.

3.1.2 Le prétraitement des données

On effectue souvent un prétraitement (un nettoyage, pour reprendre les termes de l'ECD donnés dans l'avant-propos) des données avant de les utiliser dans l'algorithme d'apprentissage. Les différents problèmes à considérer incluent :

- *Le choix des attributs de description.* Nous avons vu lors du chapitre 1 à propos de la reconnaissance de caractères comment différents choix sont possibles et peuvent avoir une influence considérable sur la difficulté d'apprendre.
- *Le traitement du bruit.* Les données disponibles sont rarement décrites parfaitement. Souvent les défauts des instruments de mesure artificiels ou humains provoquent des erreurs. Plus grave, il arrive aussi dans le cas de l'apprentissage supervisé que les réponses de l'oracle elles-mêmes soient erronées. On qualifie ces types d'erreurs de *bruit de description* et de *bruit de classification*. Finalement, il est fréquent que les données ne soient pas décrites complètement, et qu'il y ait des *valeurs manquantes* à certains attributs. C'est le cas général pour les données médicales : seul un certain nombre d'examens cliniques sont pratiqués sur chaque patient en fonction de sa pathologie, des contraintes de circonstance, etc. Ces valeurs manquantes posent souvent des problèmes difficiles à résoudre (voir par exemple le cas de l'apprentissage d'arbres de décision au chapitre 11).
- *Les données imprécises.* Les données peuvent faire l'objet de descriptions vagues : par exemple: « cet oiseau est gris ». Il faut savoir représenter de tels attributs, qui apportent une certaine information. Il faut ensuite savoir les utiliser, en particulier pour les mettre en rapport avec les connaissances sur le monde et les hypothèses.

Examinons tour à tour plus précisément ces problèmes.

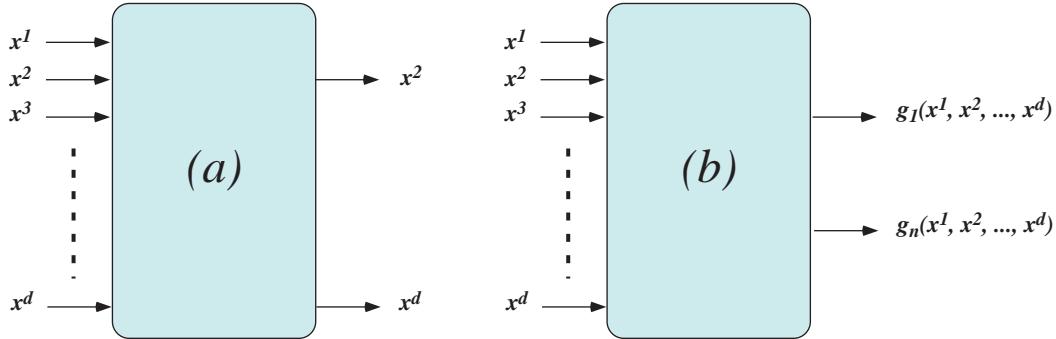


FIG. 3.2 – En (a) la sélection d’attributs retient les attributs les plus pertinents parmi les d attributs de l’espace d’entrées. En (b), l’extraction d’attributs transforme les attributs de l’espace d’entrée, ici par une fonction de combinaison g , pour en construire de nouveaux en nombre restreint.

3.1.2.1 Le choix des attributs de description des données

Généralement le choix des attributs vise à diminuer le nombre des descripteurs afin de faciliter l’apprentissage sans nuire à la qualité du résultat⁶. On distingue deux grandes approches :

- *La sélection d’attributs* consiste à éliminer les attributs les moins pertinents pour l’apprentissage. Le but est de diminuer la dimensionnalité du problème qui est à la fois une source d’imprécision et un handicap calculatoire. Si on possède une description des données par un ensemble de D attributs, le problème est de chercher un sous-ensemble de d attributs qui préserve au mieux les informations nécessaires à l’algorithme d’apprentissage.
- *L’extraction d’attributs* réduit la dimensionnalité de l’espace d’entrée en appliquant des transformations, linéaires ou non, aux attributs initiaux.

Ces deux approches sont fondées sur l’optimisation d’un certain critère J que nous ne précisons pas pour le moment. Dans le cas de la sélection d’attributs, ce critère s’applique à tous les sous-ensembles d’attributs possibles parmi les D attributs initiaux, et l’on cherche le sous-ensemble \mathcal{X}_d de dimension $d \leq D$ optimisant J :

$$J(\mathcal{X}_d) = \underset{X \in \mathcal{X}_d}{\text{Max}} J(X)$$

Dans le cas de l’extraction d’attributs, le critère traduit la qualité des transformations possibles des D attributs initiaux, et l’on cherche la transformation ϕ^* maximisant ce critère :

$$J(\phi^*) = \underset{\phi \in \Phi}{\text{Max}} J(\phi(\mathcal{X}))$$

où Φ est l’ensemble des transformations potentielles.

Le critère J est généralement basé sur une mesure de distance ou de similarité entre distributions, qui à leur tour requièrent une mesure de distance ou de similarité entre objets. Pour plus de détails sur ces mesures, nous renvoyons le lecteur à [Web99] chapitre 8.

6. Notons cependant que l’on peut imaginer au contraire de construire de nouveaux attributs qui viendront s’ajouter aux attributs initiaux. C’est ce qui est mis en jeu dans les techniques à base de fonctions noyaux, et en particulier dans la technique des séparateurs à vastes marges (SVM). Nous renvoyons le lecteur au chapitre 9 pour plus de détails.

3.1.2.2 La sélection d'attributs

Si l'on possède une description des données par un ensemble de D attributs, le problème de la sélection d'attributs consiste à chercher un sous-ensemble de d attributs qui préserve au mieux les informations nécessaires à l'algorithme d'apprentissage. Cette technique sera de nouveau évoquée un peu plus loin au paragraphe 3.1.2.3 à l'occasion de la distinction entre *filter methods* et *wrapper methods*.

Au fond, on est à peu près dans la même situation que celle du réglage des paramètres d'un algorithme (voir le paragraphe 3.4.4) : si l'on considère l'algorithme d'apprentissage comme paramétré par le sous-espace de représentation choisi, la question est de trouver le meilleur compromis entre la complexité, mesurée ici par la valeur de d , et l'efficacité, qui est la performance de l'algorithme dans l'espace de dimension réduite de D à d .

Il y a deux difficultés au problème de la sélection d'attributs :

- La première est qu'en général on recherche une méthode indépendante de tout algorithme, ceci pour ne pas faire dépendre la représentation des connaissances des choix opérationnels qui suivront. Ce n'est pas toujours le cas : on peut parfois être fixé sur le choix d'un algorithme et essayer de simplifier les données sans nuire à ses performances. Mais en principe on doit trouver une façon générique de mesurer la qualité d'un sous-ensemble d'attributs par un critère J . Ce n'est pas un problème évident. Dans le problème de classification, diverses mesures absolues de séparabilité des classes ont ainsi été définies par de nombreux auteurs ([Web99]).
- La seconde difficulté est qu'il y a $\frac{D!}{d!(D-d)!}$ sous-ensembles d'attributs de dimension donnée d et au total 2^D . Il est hors de question de mesurer sur chacun un critère de séparabilité ou la mesure de performance d'un algorithme particulier. On pourrait penser que la structure particulière de cet espace (l'ensemble des sous-ensembles d'un ensemble fini) permet d'utiliser des méthodes approximatives efficaces, mais il faut être prudent à ce sujet, comme le montre l'exemple qui suit.

Un exemple

Considérons le problème d'apprentissage de règle de classification sur un ensemble de cinq points en dimension $D = 3$ donné à la figure 3.3. Il est facile de voir que les deux classes (représentées par les symboles ● et ○) sont bien séparées, au moins sur cet ensemble d'apprentissage. Définissons un critère J , indépendant de tout algorithme, pour caractériser cette propriété. Admettons que si deux points de classes différentes sont très proches, une petite région autour d'eux va être « neutralisée », c'est-à-dire que tous les points d'apprentissage qui y sont situés seront ignorés. Le nombre de points restants est alors la valeur de J .

Puisque la séparation est parfaite en dimension 3, le critère vaut donc $J = 5$ au départ.

Si on choisit $d = 2$, les figures 3.4 montrent les projections des données dans les trois sous-espaces possibles et la valeur correspondante de ce critère (les points « neutralisés » sont entourés d'un cercle hachuré). On constate que le meilleur sous-espace est (y, z) , avec une valeur $J = 5$ pour le critère. Les sous-espaces (x, y) et (x, z) ont la valeur $J = 3$.

Pour $d = 1$, les figures 3.5 montrent que le meilleur axe est x et que les deux plus mauvais sont y et z . Par conséquent, l'algorithme glouton qui consiste à choisir la coordonnée la plus efficace seule, puis le couple le plus efficace comprenant cette coordonnée, serait en échec sur cet exemple, puisque le couple de coordonnées le plus efficace est constitué des deux coordonnées les moins efficaces.

Un grand nombre de techniques, qui relèvent de variantes adaptées de l'optimisation combinatoire, ont été proposées pour sélectionner efficacement les attributs, y compris pour fixer la valeur de d comme le meilleur compromis [Web99].

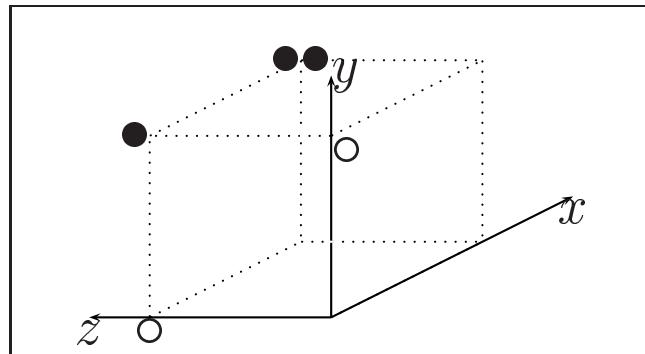
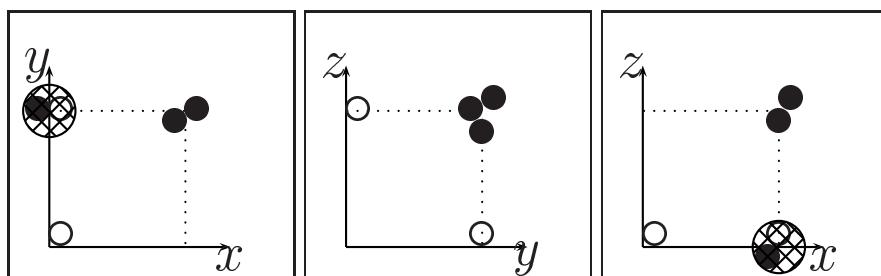
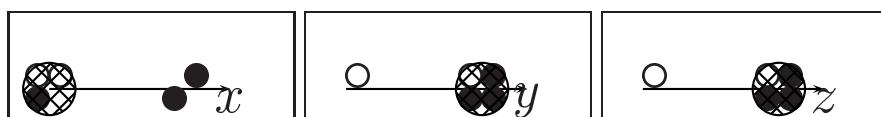


FIG. 3.3 – À trois dimensions, le critère vaut 5.

FIG. 3.4 – À deux dimensions, le meilleur sous-espace est (y, z) , avec une valeur 5 pour le critère. (x, y) et (x, z) lui donnent la valeur 3.FIG. 3.5 – À une dimension, le meilleur sous-espace est l'axe x , avec une valeur 2 pour le critère. Les axes y et z ont la valeur 1.

Si les méthodes classiques de sélection d'attributs sont aisées à trouver dans la littérature portant sur la reconnaissance des formes ou sur la fouille de données (*data mining*), il est une technique issue de l'apprentissage artificiel intéressante et peu connue qui s'applique dans le cas supervisé, en particulier quand les données sont décrites par de nombreux attributs dont la plupart ne sont pas pertinents. Il s'agit de la méthode *winnow*⁷ dite aussi de *gradient exponentiel*. Pour plus de détails sur les méthodes de gradient, on peut se reporter au chapitre 9. Une autre technique peu décrite fait appel à la théorie des ensembles approximatifs (*rough sets*) développée par Pawlak en 1985 [Paw85] et [Mod93]. L'idée est de décrire d'abord les données par un ensemble d'attributs binaires, puis de voir combien d'entre eux peuvent être retirés sans nuire à la discernabilité des données.

La méthode récente des « couvertures de Markov » (*Markov blankets*) ([KS96]) généralise cette approche à des variables discrètes. Pour chaque variable, on cherche l'ensemble (la « couverture ») des variables dont la connaissance rend inutile celle de la variable en question.

7. « Vannage », comme par exemple dans l'expression *to winnow away the chaff from the grain*, « séparer la balle du grain ».

3.1.2.3 L'extraction d'attributs

L'extraction d'attributs transforme l'espace d'entrée en remplaçant les attributs d'entrée par un ensemble plus petit correspondant si possible aux régularités sous-jacentes. On distingue souvent les approches par transformations linéaires de l'espace d'entrée de celles par transformations non linéaires. Parmi les premières, les plus usuelles sont :

- L'analyse en composantes principales, dont le but est d'identifier un petit ensemble de variables décrivant les données en minimisant la perte d'information. Cette dernière est mesurée par la variation dans l'échantillon de données, à travers une matrice de covariance ou de corrélation. Cette méthode ne prend pas en compte la classe des exemples : c'est une technique non supervisée.
- La méthode d'analyse en composantes principales communes prend au contraire en compte la classe des exemples et s'appuie sur une mesure du maximum de vraisemblance ou de l'écart aux moindres carrés.

Les méthodes d'extraction d'attributs par transformations non linéaires sont moins employées. Parmi elles figurent :

- La méthode des cartes auto-organisatrices de Kohonen, qui utilise une sorte de réseau connexionniste dans une approche non supervisée.
- Des méthodes issues des recherches sur les séparateurs à vastes marges (SVM). Nous renvoyons le lecteur intéressé à [SBE99], au chapitre 20⁸.
- L'analyse en composantes indépendantes (*Independent Component Analysis*, ICA), qui est une technique récente connaissant un grand développement. Elle s'applique dans le cas où l'on suppose que les données proviennent de plusieurs sources indépendantes, combinées par une matrice de mélange. Tandis que l'analyse en composantes principales impose seulement une indépendance des données jusqu'à l'ordre deux (mais une orthogonalité des variables), l'analyse en composantes indépendantes suppose une indépendance statistique des sources, sans contrainte d'orthogonalité. (Voir [HKO01], ou bien le chapitre 10 de [Hay99] pour une introduction).

Le chapitre 6 de [CM98] est intéressant à consulter à ce sujet, de même que les chapitres 8 et 10 de [Hay99].

Il n'est pas évident que le choix des attributs de description doive se faire préalablement à l'apprentissage. Cela suppose que le critère de choix est *a priori* correct, indépendamment de l'algorithme l'apprentissage. C'est pourquoi certaines méthodes utilisent le processus d'apprentissage lui-même pour le choix des attributs. On peut ainsi imaginer une procédure d'apprentissage (par exemple une induction d'arbre de décision) qui sélectionne les attributs les plus informatifs, puis utilise ceux-ci dans le cadre d'une autre méthode d'apprentissage. On peut aussi utiliser une procédure de sélection brutale en explorant systématiquement les sous-ensembles d'attributs possibles pour déterminer celui qui permet d'obtenir les meilleurs résultats. Cette dernière procédure est évidemment en général très coûteuse. Il faut donc distinguer les méthodes de choix *a priori* de celles qui effectuent ce choix en utilisant l'algorithme d'apprentissage. Les premières s'appellent les méthodes de filtrage (*filter methods*), les secondes méthodes d'enveloppage (*wrapper methods*), ce dernier mot pour souligner que l'apprentissage est inclus dans le processus de choix.

8. L'analyse en composante principale par fonctions noyaux.

3.1.2.4 Le traitement du bruit dans les données

Les bases de données dans lesquelles on essaie de découvrir des régularités sous-jacentes à l'aide de techniques d'apprentissage artificiel sont rarement parfaites, c'est-à-dire complètement et parfaitement décrites. Non seulement les données peuvent comporter des erreurs de description ou d'étiquetage, être imprécises, mais elles sont souvent inhomogènes, résultant de plusieurs sources rassemblées dans des contextes différents. Le plus souvent aussi, elles n'ont pas été constituées dans le but d'être analysées par une machine⁹. Il arrive aussi que des valeurs ne fournissent que des informations sur des contingences externes au problème étudié. Une banque a ainsi eu la surprise de découvrir récemment que plus de 75 % de ses clients étaient nés le 11 novembre 1911. Il était en effet plus rapide (et sans importance apparente) pour les opérateurs remplissant les fiches de taper « 111111 ». Il faut également tenir compte de conventions implicites, telles que signaler une date manquante par « 9999 », ou un poids manquant par la valeur « -1 kg ». Sans précautions, il est facile d'obtenir des résultats erronés et, ce qui est pire, sans que personne ne s'en aperçoive.

Le traitement du bruit dans les données n'est pas un problème facile à résoudre, simplement parce qu'il n'est pas facile de distinguer ce qui est le résultat d'une erreur ou d'une variation non significative d'une observation authentique. Les méthodes usuelles reposent sur des tests statistiques du niveau de pertinence. Des outils de visualisation des données peuvent être précieux dans la détection d'anomalies. Cependant rien ne remplace l'avis éclairé d'un expert et la maîtrise des phénomènes à la source des données.

Il faut aussi noter que le bruit n'est pas toujours une mauvaise chose pour l'apprentissage. Au contraire, il peut arriver que l'on introduise volontairement du bruit dans les données afin de faciliter l'apprentissage de vraies généralisations au lieu de d'apprendre par cœur les données sans en induire les régularités. L'introduction de bruit agit alors comme un facteur de régularisation (voir le chapitre 17 section 17.2.2).

3.1.2.5 La discréétisation de données continues

Certains algorithmes d'apprentissage, particulièrement les algorithmes symboliques, sont incapables de traiter directement des attributs à valeur continue. Il est nécessaire de les transformer en attributs à valeur discrète. Une autre raison pour discréteriser un attribut à valeur continue provient de ce que la distribution des valeurs peut ne pas être uniforme ou gaussienne, alors que la plupart des algorithmes en font la supposition (parfois implicite et méconnue de l'utilisateur). Il faut alors discréteriser en intervalles de distributions correspondants à des distributions uniformes ou gaussiennes.

Les méthodes de discréétisation sont nombreuses (par segmentation, par mesures d'entropie, etc.) et souvent dédiées à un contexte d'utilisation particulier. Nous renvoyons le lecteur aux publications sur le sujet, particulièrement dans le domaine de la fouille de données (*Data Mining*) : [HK01, WF99].

3.1.2.6 La description des données imprécises

L'une des méthodes les plus utilisées pour décrire des données imprécises est la logique floue. Nous renvoyons à [BM94] pour plus de détails.

9. Il arrive parfois que les valeurs manquantes soient de fait plus informatives que les autres dans la mesure où elles révèlent l'interprétation du praticien (par exemple en médecine les champs manquants suffisent souvent à déterminer le diagnostic).

3.2 L'espace des hypothèses d'apprentissage

Le chapitre 1 a souligné l'utilité de définir un espace d'hypothèses afin de ne pas avoir à représenter les concepts décrivant les données par des descriptions en extension, c'est-à-dire par des listes d'exemples. L'espace \mathcal{H} des hypothèses, défini par le langage des hypothèses $\mathcal{L}_{\mathcal{H}}$, permet le recours à une description *en intension*, compacte et permettant d'établir naturellement des liens avec les autres connaissances disponibles. Le premier problème est de savoir représenter les connaissances, donc de trouver un langage approprié au contexte et à la tâche. Le second est de savoir comment mettre en relation des hypothèses et des données. C'est ce que l'on appelle souvent le problème de l'appariement (*matching*).

3.2.1 Le problème général de la représentation des connaissances

Les représentations des connaissances en intelligence artificielle ne se font pas en langage naturel, pour des raisons évidentes. On cherche plutôt des représentations à la fois expressives et concises, permettant d'exprimer tout ce que l'on désire de manière succincte, non ambiguë, indépendante du contexte et efficace, c'est-à-dire se prêtant naturellement aux raisonnements désirés. Plusieurs types de représentations ont été développés pour répondre à ces exigences. Il est intéressant de les comparer du point de vue de l'apprentissage.

1. Quels types de régularités ou de connaissances veut-on représenter ?
 - Des catégories ou classes ou concepts.
 - Des probabilités d'appartenance à une catégorie.
 - Des ontologies, c'est-à-dire des classes organisées hiérarchiquement.
 - Des règles d'association, des réflexes.
 - Des dépendances causales.
 - Des descriptions relationnelles.
 - Des évolutions temporelles.
 - ...
2. Quelles sont les caractéristiques des entrées disponibles ?
 - Entrées perceptives brutes ou déjà prétraitées.
 - Entrées discrètes ou continues.
 - Entrées bruitées ou non.
 - Entrées correspondant à des phénomènes déterministes ou non.
 - Entrées affectées d'incertitude.
 - Entrées affectées d'imprécision.
 - Entrées « plates », telles que des vecteurs d'attributs, ou structurées par des relations et une organisation, comme des graphes.
3. Quel degré de transparence ou d'interprétabilité souhaite-t-on dans les hypothèses produites par le système ?

Ce dernier aspect est très important. Si l'on cherche seulement un système performant sur une tâche donnée, sans qu'il y ait nécessité d'interaction avec un « expert », une représentation opaque est acceptable. C'est par exemple le cas d'un système de reconnaissance de caractères ou d'identification de locuteurs sur la base d'un signal sonore. En revanche certaines applications exigent que l'utilisateur puisse examiner la connaissance produite par le système. C'est le cas d'un système de diagnostic médical et plus encore d'un système chargé de faire des recommandations thérapeutiques. Mais cela peut

aussi être utile lorsque l'expert peut aider le système à apprendre en lui transmettant des connaissances *a priori*. Encore faut-il qu'il soit alors possible de les traduire pour la machine. C'est généralement impossible avec une représentation « opaque » telle que la représentation utilisée dans les réseaux connexionnistes qui consiste en une matrice de nombres correspondant aux poids des connexions du réseau. C'est en revanche plus facile si la représentation utilise un formalisme logique.

Nous présentons maintenant les différents espaces d'hypothèses \mathcal{H} que nous allons rencontrer par la suite. Ces espaces de représentation seront décrits avec plus de précision au fur et à mesure des chapitres à venir. Pour le moment, il est seulement question de faire un tour d'horizon des représentations utilisées en apprentissage artificiel. Il est d'ailleurs intéressant de noter que toutes les techniques de représentation des connaissances utilisées en intelligence artificielle ne sont pas citées ici: certaines d'entre elles ne se prêtent pas (encore?) à l'apprentissage.

La table de la page suivante présente d'abord les qualités des différentes représentations des hypothèses en fonction des critères cités ci-dessus.

	Fonctions séparatrices	Distributions de probabilités	Fonctions état → action	Arbres de décision	Hierarchies de concepts	Réseaux bayésiens	Chaînes de Markov	Grammaires	Systèmes de règles
Concept	✓	✓	-	✓	✓	-	-	✓	✓
Classes multiples	✓	✓	-	✓	✓	-	-	-	✓
Ontologies	-	-	-	✓	✓	-	-	-	✓
Régression	-	✓	✓	✓	-	-	-	-	✓
Évolutions temporelles	-	✓	✓	-	-	-	✓	✓	-
Apprentissage non supervisé	✓	✓	✓	✓	✓	-	-	-	-
Données continues	✓	✓	✓	✓	-	-	✓	-	-
Connaissances relationnelles	-	-	✓	-	✓	✓	-	✓	✓
Degré de certitude	-	✓	✓	-	-	✓	-	✓	✓
Degré d'imprécision	-	✓	✓	-	-	✓	-	-	-
Transparence, intelligibilité	-	✓	✓	✓	✓	-	✓	✓	✓

3.2.2 La classification

3.2.2.1 Définition

L'apprentissage d'une règle de classification est l'un des thèmes de l'apprentissage artificiel le plus traité. Il y a plusieurs raisons à cela : d'abord, on sait l'aborder du point de vue des théories de l'apprentissage, la plupart du temps dans le cas de deux classes (mais on peut assez facilement généraliser à un nombre quelconque). Ensuite, un grand nombre de méthodes et d'algorithmes existent, en particulier dans le cas où l'espace de représentation est numérique. On est alors dans

le domaine classique de la reconnaissance statistique des formes (*statistical pattern recognition*). Enfin, apprendre à classer est un problème central de l'intelligence, naturelle comme artificielle.

Intuitivement, une règle de classification est un acte cognitif ou une procédure permettant d'affecter à un objet la famille à laquelle il appartient, autrement dit de le reconnaître. C'est ainsi qu'un enfant apprend à classer les animaux domestiques en « chiens » ou « chats », les plats en « salé » ou « sucré », etc. Par analogie, les ordinateurs de bureau qui reconnaissent l'écriture manuscrite ont appris (grâce à un programme d'apprentissage automatique) des règles pour distinguer les signes tracés ; d'autres programmes savent classer des sons, des signaux biomédicaux, etc. Toutes les procédures qui simulent des fonctions perceptives doivent évidemment posséder des capacités de généralisation, c'est-à-dire être munies de la faculté d'induction, sans quoi elles ne seraient capables de reconnaître que les exemples qui ont servi à les entraîner.

3.2.2.2 Classe, concept

Définition 3.1 (exemple)

Un exemple est un couple (\mathbf{x}, \mathbf{u}) , où $\mathbf{x} \in \mathcal{X}$ est la description ou la représentation de l'objet et $\mathbf{u} \in \mathcal{U}$ représente la supervision de \mathbf{x} . Dans un problème de classification, \mathbf{u} s'appelle la classe de \mathbf{x} et appartient à un ensemble $\mathcal{C} = \{\omega_1, \dots, \omega_C\}$. C désigne le nombre de classes possibles pour un objet.

C doit être fini et en pratique petit pour que l'on puisse réellement parler de classification. Des exemples de classes sont : les sons du langage, l'alphabet, les espèces des oiseaux, un diagnostic médical, la présence ou l'absence d'une propriété pour un objet (par exemple qu'une carte à jouer soit un « honneur »), etc.

Dans le cas où $C = 2$, il est usuel de considérer que l'on fait l'apprentissage d'un *concept*, c'est-à-dire du partage de l'espace de représentation en deux parties, l'une où le concept est vérifié, l'autre où il est invalidé. Dans ce cas, on note¹⁰ en général $\mathcal{C} = \{VRAI, FAUX\}$ et on appelle *contre-exemples* les données classées *FAUX* (on garde le mot d'*exemples* pour les autres).

Il est à noter que le cas $C = 1$ est presque équivalent au précédent, puisqu'il s'agit d'apprendre aussi un concept, mais à partir seulement d'exemples ; en pratique, cependant, les algorithmes seront différents.

Par exemple, un enfant apprend sa langue maternelle avec un « algorithme » de généralisation où le rôle des contre-exemples est faible. En revanche, il classe les matières d'enseignement en celles qu'il aime et celles qu'il n'aime pas à partir d'une base d'apprentissage composée d'exemples des deux cas.

3.2.2.3 Les fonctions séparatrices entre classes

Au lieu d'essayer d'approcher directement la fonction de classification cible $f : \mathcal{X} \rightarrow \{\omega_1, \dots, \omega_C\}$ par une règle de classification, il est souvent plus facile de transformer l'espace des classes en celui des fonctions séparatrices.

Définition 3.2 (fonction séparatrice)

Une fonction séparatrice, ou fonction de décision $\Upsilon_{ij} : \mathcal{H} \rightarrow \mathbb{R}$ entre la classe ω_i et la classe ω_j est telle que $\Upsilon_{ij}(\mathbf{x}) \geq 0$ pour tous les objets \mathbf{x} que la fonction cible affecte à la classe ω_i et $\Upsilon_{ij}(\mathbf{x}) \leq 0$ pour tous les objets qu'elle affecte¹¹ à la classe ω_j .

10. Parfois $\mathcal{C} = \{+, -\}$, ou $\mathcal{C} = \{1, 0\}$.

L'espace de l'apprentissage devient alors un ensemble d'hypothèses constitué de fonctions séparatrices. Ces fonctions peuvent être de natures extrêmement variées : par exemple des hyperplans (voir le chapitre 9), ou calculées à partir de réseaux connexionnistes multicouche (voir le chapitre 10) ou de densités de probabilité (voir le chapitre 14), etc.

Dans la définition ci-dessus, on ne considère que le signe de la fonction de décision pour décider de la région d'appartenance de l'entrée \mathbf{x} (voir figure 3.6 (a)). On parle souvent dans ce cas de *fonctions séparatrices à seuil*. Dans le cas où il y a plus de deux classes, on peut combiner plusieurs fonctions de décision permettant ainsi une division de \mathcal{X} en plusieurs régions (voir figure 3.6 (b)). On y reviendra au chapitre 9.

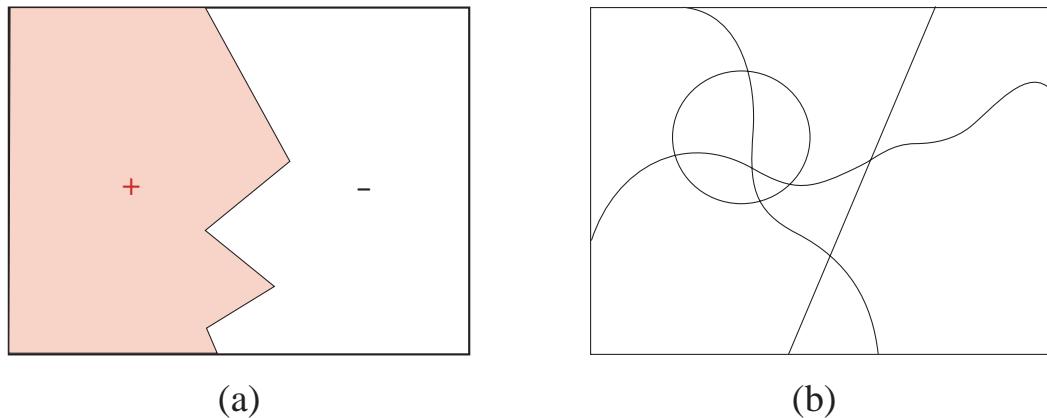


FIG. 3.6 – *Classification par fonctions séparatrices.* En (a) la fonction séparatrice détermine deux classes suivant le signe de la sortie de la fonction. En (b), une classification pour plus de deux classes est obtenue par la combinaison de plusieurs fonctions séparatrices.

Si en plus de son signe, on considère aussi la valeur de la sortie de la fonction de décision, il devient possible d'interpréter cette dernière comme une mesure de confiance dans la décision, selon l'idée naturelle que plus la forme d'entrée est « éloignée » de la frontière, plus son appartenance à la classe désignée est peu susceptible d'être remise en cause. Nous verrons que cette observation de bon sens est à l'origine d'un renouveau très fort pour l'utilisation de ces fonctions de décisions (voir les séparateurs à vastes marges dans le chapitre 9).

En dehors de leur simplicité conceptuelle et pratique évidente, les fonctions séparatrices permettent de mettre en œuvre naturellement un appariement partiel entre entrée et hypothèse. En effet, les fonctions séparatrices peuvent se concevoir comme une sorte de produit scalaire défini sur $\mathcal{X} \times \mathcal{H}$. Ainsi, dans le cas du perceptron, déjà rencontré au cours du chapitre 2, la fonction de décision est définie par :

$$\mathbf{w}^T \mathbf{x} \quad \begin{cases} \geq 0 \\ < 0 \end{cases} \implies \mathbf{x} \in \begin{cases} \omega_1 \\ \omega_2 \end{cases} \quad (3.1)$$

en considérant le vecteur de description des observations augmenté $\mathbf{x}^T = (1, x_1, x_2, \dots, x_d)$ et le vecteur poids \mathbf{w} augmenté du seuil w_0 : $\mathbf{w}^T = (w_0, w_1, w_2, \dots, w_d)$.

Cette faculté d'appariement partiel dans lequel c'est l'« alignement » entre l'entrée et l'hypothèse qui décide de la classe de l'entrée est une propriété très intéressante qui n'est pas

11. L'affectation pour $\Upsilon_{ij}(\mathbf{x}) = 0$ se fait en général arbitrairement.

aussi facile à mettre en œuvre dans les formalismes logiques par exemple. C'est une des raisons de la popularité des fonctions de décision.

3.2.3 La régression

La régression concerne le cas où \mathcal{H} est un ensemble de fonctions h à valeurs réelles. Une généralisation, la régression multidimensionnelle, est l'apprentissage d'une hypothèse $h : \mathcal{X} \rightarrow \mathbb{R}^n$. On cherche donc à apprendre une fonction à partir d'un ensemble de points et des valeurs que prend cette fonction sur ces points. Il n'y a pas de contre-exemples dans un tel problème d'apprentissage¹².

Il sera en particulier question de régression quand nous verrons l'apprentissage par renforcement, au chapitre 16.

3.2.4 Les distributions de probabilités

Au lieu de délimiter des frontières de décision sur l'espace \mathcal{X} , on peut y définir des distributions de probabilités. Chacune de ces distributions est associée à une classe et détermine la probabilité qu'un objet $x \in \mathcal{X}$ appartienne à la classe (voir figure 3.7). Pour qu'il soit aisément de manipuler ces distributions et que l'on puisse contrôler leur pouvoir de généralisation (cf. chapitre 2), elles sont généralement prises au sein de familles paramétrées de distributions, par exemple des fonctions gaussiennes. Nous y reviendrons au chapitre 14.

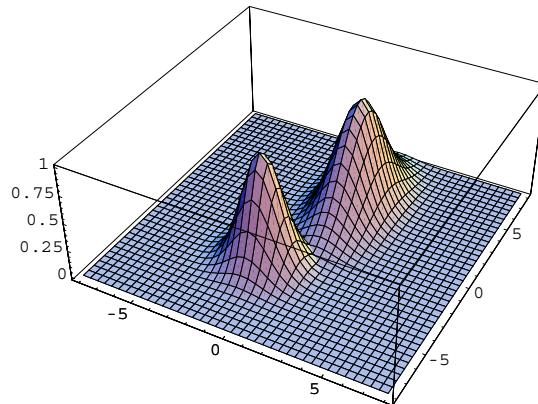


FIG. 3.7 – Deux distributions de probabilité correspondant à deux classes d'objets.

3.2.5 Les arbres de décision

Lorsque les exemples s'expriment comme des vecteurs d'attributs valeurs, et particulièrement quand ces attributs sont à valeurs discrètes, il est commode de décrire les concepts par des *arbres de décision* comme celui de la figure 3.8. Un arbre de décision prend la description d'un exemple en entrée et lui associe une classe. Chaque nœud de l'arbre correspond à une question portant sur un attribut¹³.

De la sorte, en suivant une séquence de nœuds et de branches depuis la racine de l'arbre jusqu'à une feuille, on raffine progressivement la description des exemples concernés jusqu'à

-
- 12. On peut aussi voir la régression comme un problème de classification généralisé, dans lequel le nombre C de classes serait infini.
 - 13. Ces attributs peuvent aussi être numériques, comme on le verra au chapitre 11. Dans ce cas, on les compare à un seuil et chaque branche est associée à une valeur ou à un intervalle de valeurs possibles pour cet attribut.

obtenir une description correspondant, si tout va bien, aux objets d'une classe. Chaque branche correspond à une conjonction de conditions sur les attributs décrivant les exemples.

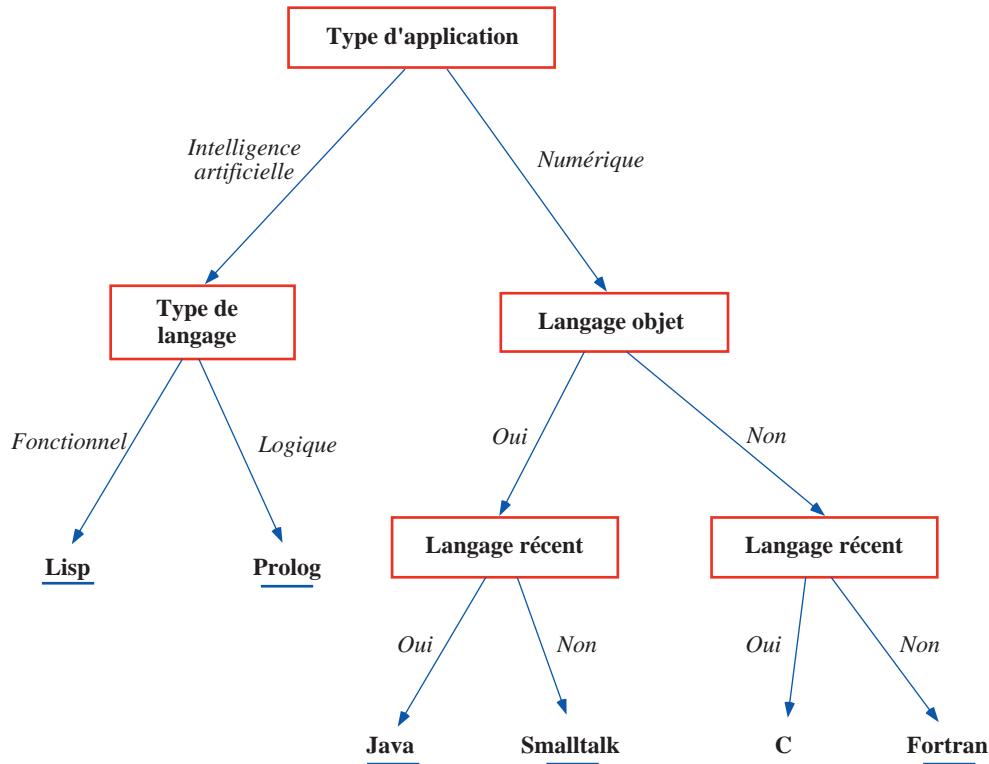


FIG. 3.8 – Un exemple d'arbre de décision. Chaque nœud rectangulaire correspond à une question. Chaque flèche correspond à une réponse possible. Ici toutes les questions sont binaires, mais ce n'est pas nécessairement le cas.

Par exemple, dans la figure 3.8, une interprétation est la suivante: << Type d'application = intelligence artificielle & type de langage = logique >> → Prolog)

L'ensemble des branches correspond ainsi à un ensemble de règles d'association décrivant les classes. Le langage défini par les arbres de décision est équivalent à la logique des propositions, chacun des tests étant une variable booléenne. Toute fonction booléenne peut être exprimée par un arbre de décision.

En revanche, un arbre de décision ne peut pas exprimer un concept *relationnel* comme :

$$\exists x \text{ même-couleur}(x, y) \& \text{envergure}(x, e1) \& \text{envergure}(y, e2) \& \text{plus-petit}(e1, e2)$$

dont la signification est : « les oiseaux x de même couleur qu'un oiseau donné y mais d'envergure inférieure » : ce type de concept appartient à la logique des prédictats (voir le chapitre 5).

Par ailleurs, si certaines fonctions s'expriment de manière économique à l'aide d'arbres de décision, d'autres ne sont pas adaptées à cette représentation. Par exemple la *fonction parité* qui retourne 1 si et seulement si un vecteur booléen si un nombre pair d'attributs valent 1 s'exprime par un arbre très complexe.

Nous approfondirons cette manière de représenter les concepts au chapitre 11.

3.2.6 Les hiérarchies de concepts

Les arbres de décision introduisent l'idée de hiérarchie sur les attributs, mais pas sur les concepts. Les attributs placés plus près de la racine sont en quelque sorte plus importants que

les attributs plus éloignés.

Il peut être souhaitable d'exprimer explicitement une hiérarchie dans le langage des concepts. C'est le cas par exemple pour les taxonomies de la classification biologique : le concept de **rapace** est situé en dessous du concept **oiseau** et plus haut que **faucon**. Il peut également être intéressant de disposer de relations d'héritage entre un concept et ses sous-concepts. De nombreux types de représentations de concepts peuvent donner lieu à des organisations hiérarchiques, ou ontologies pour peu que l'on explicite les liens de hiérarchie. Cependant de telles organisations sont alors souvent artificielles.

L'idéal est que l'apprenant lui-même construise la hiérarchie et soit prêt à la modifier si de nouvelles informations en indiquent l'utilité. Il existe peu de systèmes d'apprentissage aptes à de telles constructions. On les rencontre généralement en apprentissage non supervisé, quand c'est au système de découvrir des classes dans l'environnement. Le chapitre 15 fournit des précisions à ce sujet.

3.2.7 Les réseaux bayésiens et les modèles graphiques

De nombreux types de dépendances peuvent être représentés à l'aide de structures probabilistes. Lorsque les dépendances et les indépendances entre les variables aléatoires sont explicitées dans un graphe, on souligne cette transparence en parlant de *modèles graphiques* (on trouve aussi les termes de « réseaux de croyance » (*belief networks*), « réseaux causaux » (*causal networks*), « diagrammes d'influence » (*influence diagrams*)). La base des calculs effectués dans ces structures est la formule de révision des probabilités de Bayes et pour cette raison ils sont également appelés *réseaux bayésiens*.

La figure 3.9 montre un exemple d'un tel réseau. Les variables sont associées aux nœuds du réseau et les liens manquants entre les nœuds indiquent une certaine indépendance entre ces nœuds (les définitions précises seront données dans le chapitre 12). Les liens entre nœuds *s* sont dirigés, pour indiquer des dépendances causales ou temporelles. Lorsque les liens sont symétriques ou non dirigés, on parle de champs de Markov aléatoires (*random Markov fields*).

Ici les deux variables FN et ZO jouent le rôle de variables causales, CP pouvant découler de FN et/ou de ZO, tandis que SA ne dépend, selon ce réseau, que de FN. Les variables FN et ZO sont affectées de leur probabilité *a priori*, tandis que les variables SA et CP sont associées à des matrices de probabilités conditionnelles indiquant leur dépendance sur les variables FN et ZO.

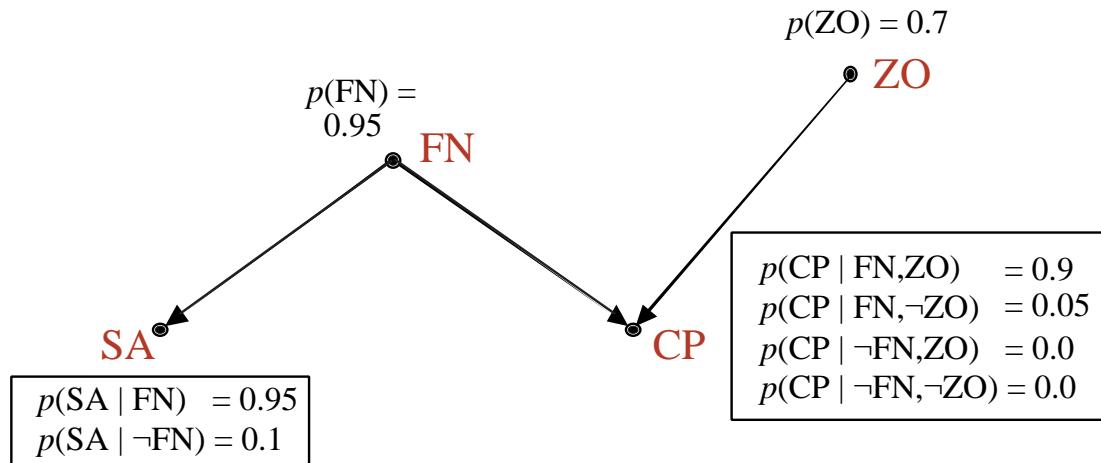


FIG. 3.9 – Un exemple de réseau bayésien.

L'apprentissage peut alors consister à trouver une structure de dépendances entre variables (le graphe) ou à estimer les probabilités conditionnelles définissant ces dépendances. Le chapitre 12 traite en particulier de l'apprentissage de réseaux bayésiens.

3.2.8 Les chaînes de Markov et les modèles de Markov cachés

À strictement parler, les chaînes de Markov ne sont qu'un cas particulier de modèles graphiques tels qu'ils viennent d'être décrits. Elles représentent en effet les dépendances temporelles dans une séquence de variables S_0, \dots, S_t, \dots . Lorsque chaque variable ne dépend que de la variable précédente dans la séquence, on dit que l'on a une *chaîne de Markov*:

$$P(S_t|S_0, \dots, S_{t-1}) = P(S_t|S_{t-1})$$

Intuitivement, cela peut être interprété comme le fait que le futur ne dépend que de l'état présent, ou du moins que celui-ci possède suffisamment d'informations pour qu'il ne soit pas utile de considérer le passé.

Le formalisme des chaînes de Markov est particulièrement adapté pour la représentation de séquences, qu'elles soient de nature temporelle, comme par exemple des cours boursiers ou un signal acoustique, de nature spatiale, comme par exemple une chaîne d'ADN, ou d'autres types de dépendances linéaires.

Une généralisation des chaînes de Markov s'appelle les *modèles de Markov cachés* (*Hidden Markov Models* ou HMM). Formellement, un modèle de Markov caché (d'ordre un) est un modèle génératif de séquences défini par un ensemble d'états, un alphabet discret de symboles, une matrice de probabilités de transitions entre états et une matrice de probabilité d'émissions de chaque symbole de l'alphabet à partir de chaque état. Le système évolue aléatoirement d'un état à l'autre suivant les probabilités de transition en émettant des symboles de l'alphabet.

Seuls les symboles émis sont observables, et non les transitions entre états, qui sont internes au modèle. La séquence d'états est donc une séquence de variables cachées ou latentes expliquant les observations.

Trois types de questions au moins peuvent se poser lorsque l'on représente une séquence par un modèle de Markov : quelle est la probabilité d'observation de telle séquence étant donné tel modèle? (Question relative à la vraisemblance). Quelle est la séquence d'états la plus probable dans le modèle de Markov sachant que telle séquence de symboles a été observée? (Question relative au décodage). Finalement, en supposant que les paramètres de transition et d'émission ne soient pas parfaitement connus, comment leurs valeurs devraient être estimées ou révisées à la lumière des séquences de symboles observées? (Question relative à l'apprentissage).

Le chapitre 13 est consacré aux méthodes d'apprentissage adaptées à ce formalisme des modèles de Markov cachés.

3.2.9 Les grammaires

Quand on a affaire à des séquences d'éléments d'un ensemble nominal, souvent appelé un alphabet dans ce cas, le concept à apprendre doit séparer l'espace de toutes les séquences possibles en deux. C'est ce que fait une grammaire formelle: un compilateur de langage de programmation est un programme qui répond à la question: « est-ce que le programme que l'on vient de me soumettre est correct du point de vue de ma syntaxe? ». Par exemple, un compilateur du langage C répond *VRAI* à la séquence :

```
#include <stdio.h> #include <math.h> int N;double x,res; main()
{N=0;while (N<21){fprintf(stdout, " %f      %f\n",n);N = N+1;}}
```

Il répond *FAUX* à celle-ci :

Je sais programmer en C.

Naturellement, une grammaire du français répondrait exactement le contraire.

Les modèles grammaticaux que l'on sait apprendre ne sont pas en réalité aussi complexes. On verra au chapitre 7 comment on peut par inférence grammaticale généraliser des ensembles de séquences, en particulier sous la forme d'automates finis.

3.2.10 Les formalismes logiques

La logique des propositions

Si on dispose de l'ensemble des exemples et contre-exemples suivants, décrits par des attributs binaires :

vole	a des plumes	pond des œux	oiseau	classe
VRAI	VRAI	VRAI	VRAI	oie
FAUX	FAUX	VRAI	FAUX	ornithorynque
VRAI	FAUX	FAUX	FAUX	rhinolophe
VRAI	VRAI	VRAI	VRAI	cygne

On peut par exemple induire le concept ci dessous :

$$h = [(vole) \wedge (a \text{ des plumes}) \Rightarrow (oiseau)]$$

Ce concept est écrit dans le langage de la logique des propositions, ou logique booléenne ou encore logique d'ordre 0. Il est *VRAI* pour tous les exemples. Mesurons sa valeur sur les objets suivants :

vole	a des plumes	pond des œux	oiseau	classe	valeur de h
VRAI	VRAI	VRAI	VRAI	moineau	VRAI
FAUX	VRAI	VRAI	VRAI	autruche	FAUX

Le premier objet est représenté correctement par le concept, mais pas le second. À supposer que l'on veuille apprendre le concept « oiseau », la généralisation réalisée ne serait donc ici pas parfaite.

Ce langage de représentation des hypothèses est comme on le voit particulièrement adapté aux exemples représentés par des vecteurs binaires, ou dont la représentation naturelle peut facilement être transformée en vecteurs binaires¹⁴. Le chapitre 11 traitera en partie de l'apprentissage de ce type d'hypothèses, sous la forme particulière d'arbres de décision. Il en sera également question au chapitre sur l'espace des versions (chapitre 4).

La représentation par attribut-valeur

La logique des propositions peut s'étendre en remplaçant les valeurs binaires des attributs par des valeurs nominales ou hiérarchiques. Les exemples se représentent alors de la manière suivante :

couleur	forme	nombre de pieds	classe
rouge	hexagonale	3	tabouret
jaune	carrée	4	tabouret
vert	ronde	4	table
jaune	ovale	6	table

14. Par exemple, un attribut continu peut être transformé en attribut binaire par comparaison à un seuil.

Le langage attribut-valeur dans lequel on représente le concept appris est semblable à celui de la logique des propositions : on y utilise aussi des conjonctions et des disjonctions, mais sur des couples (*attribut, valeur*). Chaque attribut nominal ou hiérarchique (par exemple *couleur*, voir la figure 3.1) prend sa valeur dans un ensemble de définition fini, éventuellement partiellement ordonné, comme $\{\text{rouge}, \text{vert}, \text{jaune}, \text{bleu}\}$, avec $\text{couleur} - \text{chaude} = \{\text{rouge}, \text{jaune}\}$. Un concept *tabouret* appris dans le langage attribut-valeur pourrait être par exemple :

$$[\text{couleur} = \text{couleur} - \text{chaude}] \wedge ([\text{forme} = \text{carrée}] \vee [\text{forme} = \text{hexagonale}])$$

L'intérêt des langages par attribut-valeur est celui des langages typés par rapport aux langages non typés : ils permettent un contrôle plus facile des inférences.

La logique des prédictats

Supposons que nous disposions des données ci-dessous, dont la signification formelle sera donnée au chapitre 5. Pour le moment, nous pouvons en rester à l'interprétation suivante : le numéro 1 ou 2 représente un individu. La relation $Fille(1, 2) = VRAI$ s'interprète comme : *l'individu 2 est une fille de l'individu 1*. Les autres relations ont une signification naturelle.

$$\begin{array}{llll} \text{Nom}(1) = \text{Ève} & \text{Mère}(1) = \text{Marie} & \text{Père}(1) = \text{Max} & \text{Homme}(1) = FAUX \\ \text{Nom}(2) = \text{Max} & \text{Mère}(2) = \text{Adèle} & \text{Père}(2) = \text{Max} & \text{Homme}(2) = VRAI \\ \text{Fille}(1, 2) = VRAI \end{array}$$

D'autre part, le programme d'apprentissage dispose de connaissances *a priori*, comme :

$$\forall x \text{ } \text{Homme}(x) = VRAI \iff \text{Femme}(x) = FAUX$$

$$\forall x \text{ } \text{Homme}(x) = FAUX \iff \text{Femme}(x) = VRAI$$

À partir de ces exemples et de cette *théorie du domaine*, un programme de généralisation en *logique du premier ordre* peut apprendre le concept :

$$\forall x \forall y \text{ } (\text{Père}(y) = x) \wedge (\text{Femme}(x) = VRAI) \Rightarrow (\text{Fille}(x) = y)$$

La différence avec l'apprentissage en logique des propositions est importante : cette formule est gouvernée par des quantificateurs \forall (*quelque soit*) et \exists (*il existe*) qui sont hors du langage booléen. La généralisation réalisée est par conséquent beaucoup plus radicale et profonde (elle est aussi plus difficile à faire). L'apprentissage de tels concepts s'appelle la programmation logique inductive. Il sera développé au chapitre 5.

Les logiques de description

Les logiques de description forment une famille de formalisme de représentation des connaissances dédiée principalement à la gestion automatique de la définition de concepts (ensemble d'individus) et de raisonnement sur ces concepts. Les concepts sont partiellement ordonnés dans une base de connaissances organisée en taxonomie par une relation de subsumption. Basés sur des calculs de subsumption, les principaux mécanismes d'inférence déductive sont la *classification de concept*, qui consiste à insérer automatiquement un concept défini à la place la plus spécifique dans la taxonomie, et la « reconnaissance d'instances » qui consiste à trouver pour un individu donné tous les concepts dont il est instance. De nombreux systèmes de représentation ont été définis dans ce cadre, citons ici le système CLASSIC [BPS94].

3.3 La recherche dans l'espace des hypothèses

Une fois choisi l'espace \mathcal{H} des hypothèses, apprendre revient essentiellement à chercher la meilleure hypothèse dans \mathcal{H} . À condition de disposer d'une évaluation de chaque hypothèse par une fonction de coût, on peut donc considérer l'apprentissage comme un problème d'optimisation : il s'agit de trouver l'élément de \mathcal{H} pour lequel la fonction de coût prend la valeur maximale.

L'ensemble des techniques développées pour résoudre les problèmes d'optimisation est très vaste. Cette section vise à jalonner le panorama des méthodes relevant des problèmes spécifiques qui peuvent se poser en apprentissage, sans avoir l'ambition de l'exhaustivité (le lecteur pourra se reporter par exemple à [PS82] et [CGH96]).

L'optimisation consiste donc en principe à identifier une solution optimale vis-à-vis de la fonction de coût, dans un ensemble de solutions. Cependant, trouver une solution optimale est en général irréalisable, et l'on doit se contenter d'approches *locales*, opérant par itérations et dont le résultat, un *optimum local*, dépend du point initial. On distingue les *méthodes d'exploration directes*, dans lesquelles l'exploration se fait par évaluation effective des solutions du voisinage, des *méthodes indirectes* dans lesquelles l'exploration est guidée par des propriétés de la fonction de coût (comme les dérivées par exemple). Les méthodes d'optimisation dépendent du type de solution cherchée, de la fonction de coût à optimiser, des contraintes éventuelles à satisfaire et des propriétés de l'espace à explorer.

3.3.1 Caractérisation de l'espace de recherche

L'une des méthodes d'optimisation les plus générales consiste à utiliser un algorithme de recherche locale. Étant donnée la définition d'un voisinage dans l'espace des solutions, l'idée est d'améliorer au fur et à mesure une solution provisoire en la remplaçant par une solution meilleure et située dans son voisinage. Cette notion de voisinage est par conséquent centrale, puisqu'elle offre la possibilité d'explorer rationnellement l'espace des hypothèses.

En ce qui concerne ces derniers, il faut distinguer les *espaces discrets* des *espaces continus*. On rencontre les premiers principalement dans les problèmes d'apprentissage impliquant des représentations symboliques des hypothèses (par exemple des grammaires). Les espaces continus sont liés à des représentations numériques des hypothèses (par exemple des réseaux connexionnistes).

Dans les espaces discrets, les éléments du voisinage sont définis par des opérateurs de modification syntaxique. Il est possible de les énumérer et d'explorer exhaustivement le voisinage d'une hypothèse. Dans les espaces continus, les éléments du voisinage sont généralement définis par des opérateurs différentiels et les directions de recherche sont choisies à partir du calcul de dérivées de la fonction de coût au voisinage du point courant.

Il existe également des problèmes d'optimisation dans lesquels les solutions ne sont définies que pour des domaines de validité des différentes variables en jeu. Ce sont en général les problèmes dits de « satisfaction de contraintes ».

3.3.2 Caractérisation des fonctions de coût

3.3.2.1 L'optimisation d'une fonction numérique de coût

On dispose d'une fonction dite *de coût* $c(h)$, définie pour chaque hypothèse h . Une hypothèse dépend elle-même d'un ensemble de variables. Il s'agit de trouver des valeurs à ces variables telles que la fonction coût soit minimale. Cette fonction est en général réelle positive, par exemple le risque empirique si on se place dans le principe d'apprentissage *ERM*. Les variables sont dans ce

cas les paramètres de l'hypothèse à apprendre : par exemple les poids d'un réseau connexionniste. Deux grandes familles de techniques se rencontrent :

1. **L'optimisation sans contrainte.** Le problème est de trouver le minimum de la fonction de coût $c(h)$ définie sur l'ensemble complet des solutions \mathcal{H} .
 - Si $c(h)$ est linéaire à valeur réelle et dérivable par rapport aux variables, ce sont en particulier les techniques de la *programmation linéaire* qui peuvent s'appliquer.
 - Si $c(h)$ n'est pas linéaire en fonction des variables, mais est à valeur réelle et dérivable par rapport à ces variables, alors les méthodes de la programmation non linéaire peuvent s'utiliser. Toutes les méthodes d'optimisation locales par gradient rentrent notamment dans cette catégorie.
 - Il est intéressant de distinguer le cas particulier où la fonction $c(h)$ est convexe par rapport aux variables qui définissent h . En effet, dans ce cas il n'y a qu'un seul optimum de la fonction, qui est évidemment l'optimum global. On ne risque donc pas de s'égarer dans un optimum local.
2. **L'optimisation avec contraintes.** Il s'agit de trouver l'optimum d'une fonction objectif sous certaines contraintes portant sur les variables de décision. Ces contraintes qui définissent le domaine de validité des variables peuvent prendre la forme d'égalités (hypersurfaces dans le domaine de définition des variables) ou d'inégalités (régions admissibles). Les contraintes peuvent être linéaires ou non. Le cas des contraintes linéaires sera abordé à l'occasion de l'apprentissage de séparateurs à vastes marges (SVM) au chapitre 9 mettant en jeu la méthode des multiplicateurs de Lagrange.

3.3.2.2 L'optimisation d'une fonctionnelle

Il peut s'agir aussi de chercher une fonction dont une certaine propriété est optimale. Par exemple, le problème peut être de trouver une trajectoire de longueur minimale, ou parcourue en un temps minimal, ou encore de trouver une séquence d'états de probabilité maximale dans une chaîne de Markov. Plusieurs techniques sont applicables. Si l'espace est discret, on peut envisager les méthodes de recherche dans les graphes développées en intelligence artificielle, comme les algorithmes par séparation et évaluation (*branch and bound*) et particulièrement l'algorithme A^* ([Nil98]). On peut aussi espérer pouvoir appliquer les techniques de la *programmation dynamique*.

3.3.3 Les méthodes d'optimisation

3.3.3.1 L'optimisation par gradient dans un espace continu

On se place ici dans le cas où l'espace des hypothèses \mathcal{H} est continu et où on cherche à minimiser le risque empirique (principe *ERM*). On peut le supposer dérivable¹⁵ par rapport aux paramètres qui caractérisent l'hypothèse. En pratique, \mathcal{H} est toujours assimilable à \mathbb{R}^d . Notons \mathbf{w} le vecteur de \mathbb{R}^d caractérisant une hypothèse à l'étape t . Le risque empirique de l'hypothèse courante peut alors se noter¹⁶ $R_{Emp}(\mathbf{w}^t)$.

Pour approcher l'hypothèse cherchée, celle qui minimise le risque empirique, l'un des algorithmes d'apprentissage les plus simples consiste à ajuster le vecteur \mathbf{w} par une procédure de descente de gradient afin d'optimiser la mesure de performance de $R_{Emp}(\mathbf{w}^t)$ sur \mathcal{S} . On obtient

15. En toute rigueur, cela supposerait un nombre infini d'exemples... On se contente d'une approximation naturelle.

16. Nous emploierons un peu plus loin la notation $R_{Emp}(h_{algo, \mathcal{S}, \mathcal{H}}^t)$, mais pour l'instant celle-ci est suffisante.

alors une séquence de vecteurs $\mathbf{w}^0, \mathbf{w}^1, \dots, \mathbf{w}^t$ obéissant à l'équation de récurrence :

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \alpha \frac{\partial R_{Emp}(\mathbf{w})}{\partial \mathbf{w}} \Big|_{\mathbf{w}^t} \quad (3.2)$$

où α est un réel positif que l'on appelle le *pas d'apprentissage* et $\frac{\partial R_{Emp}(\mathbf{w})}{\partial \mathbf{w}}|_{\mathbf{w}^t}$ est la dérivée partielle du critère $R_{Emp}(\mathbf{w})$ par rapport au paramètre \mathbf{w} à l'étape \mathbf{w}^t .

Cette équation exprime l'algorithme suivant : pour choisir la prochaine valeur \mathbf{w}^{t+1} , on retranche à \mathbf{w}^t une quantité proportionnelle à la valeur au point \mathbf{w}^t de la dérivée du risque empirique par rapport à \mathbf{w} .

Dans l'espace IR, l'illustration de la méthode et de ses faiblesses est donnée sur la figure 3.10. La fonction à optimiser est notée f et l'unique valeur cherchée est notée u^* .

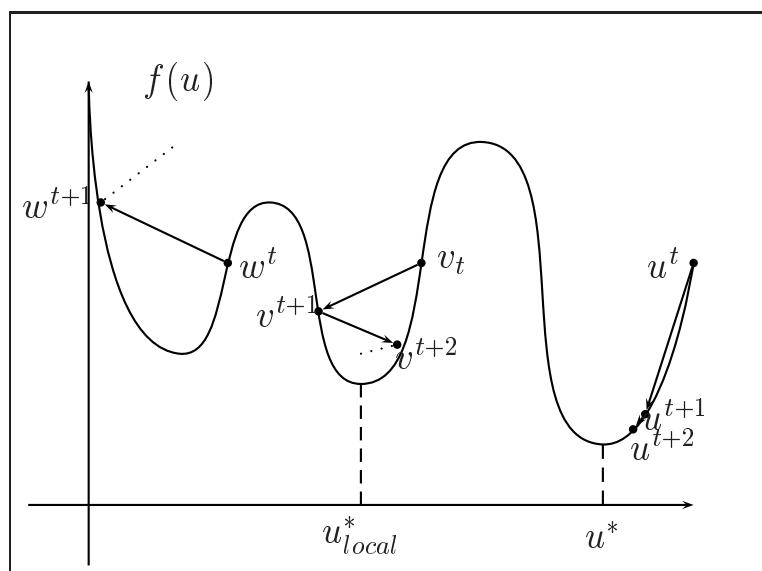


FIG. 3.10 – Une illustration de la méthode de descente de gradient.

On y voit trois cas caractéristiques :

- Si u^t se trouve dans la partie droite du graphe, le gradient $\frac{\partial}{\partial u} f(u)|_{u^t}$ est positif et grand. Par conséquent, en prenant

$$u^{t+1} = u^t - \alpha \frac{\partial}{\partial u} f(u) \Big|_{u^t}$$

avec α réel positif, on obtiendra u^{t+1} au point indiqué. À cet endroit, le gradient est plus petit, quoique toujours positif. Le point suivant u^{t+2} , calculé avec la même valeur de α , sera donc plus proche de u^{t+1} que u^{t+1} l'était de u^t . On assiste de la sorte à la convergence de la suite vers la valeur minimale u^* cherchée.

- Construisons maintenant une autre suite à partir du point v^t , avec une valeur supérieure pour α . Cette suite converge aussi, mais il y a deux différences avec le cas précédent :
 - La suite n'est plus strictement croissante ou décroissante, mais converge par valeurs alternativement supérieures et inférieures à la valeur finale.
 - La valeur atteinte n'est pas le minimum u^* , mais une u_{local} . Cette « erreur » de convergence vers un *minimum local* se rencontre très souvent en pratique.
- À partir du point noté \mathbf{w}^t , en prenant α encore plus grand, la suite va diverger.

3.3.3.2 Les méthodes locales pour les problèmes d'optimisation combinatoire

Les problèmes d'optimisation combinatoire font partie des mathématiques discrètes. Leur objet est de trouver une solution optimale (locale ou de préférence globale) dans un ensemble fini mais souvent extrêmement vaste de solutions potentielles sur lequel est définie une fonction de coût. La difficulté de ces problèmes est qu'ils sont souvent *NP*-difficiles, ce qui signifie en pratique qu'il est admis¹⁷ qu'il n'existe pas d'algorithme capable de les résoudre en temps polynomial en fonction de leur taille. C'est pourquoi ont été développés des algorithmes sous-optimaux fondés sur l'utilisation de méta-heuristiques de recherche¹⁸. Une méta-heuristique est donc une stratégie d'exploration de l'espace des solutions. Les méthodes locales explorent l'espace localement. L'idée est de *définir un voisinage*, c'est-à-dire d'associer à chaque solution un ensemble de solutions voisines, puis d'améliorer la solution courante en la remplaçant de manière itérative par (généralement) une meilleure solution dans son voisinage.

Les méthodes de recherche locale, classiques en recherche opérationnelle et en intelligence artificielle (en particulier toutes les techniques d'exploration de graphes, comme A^*), ont connu récemment des développements spectaculaires grâce au concept d'exploration stochastique de l'espace. Les méthodes de recuit simulé, les méthodes tabou et les méthodes d'optimisation par algorithmes génétiques sont ainsi devenues centrales dans la panoplie du parfait praticien de l'optimisation.

Sans entrer dans les détails, nous rappelons brièvement les bases de quelques méthodes dans la suite.

Les méthodes de gradient dans un espace discret

Dans les espaces d'hypothèses discontinus, le principe de la descente de gradient doit être adapté pour la simple raison que la notion de dérivabilité n'existe plus. Mais il est possible d'en conserver le principe, en définissant une notion de voisinage. L'optimisation se déroule alors selon le principe suivant : en cours de recherche, on se trouve sur une certaine hypothèse h . Si elle est la meilleure dans son voisinage, elle sera considérée comme localement optimale. Sinon, on choisira celle dans le voisinage qui fait gagner le plus sur le critère à optimiser (en général l'erreur empirique).

Dans le paragraphe suivant, nous formalisons un peu plus cette approche discrète de l'optimisation par gradient dans le cas où l'espace que l'on explore peut être mis sous la forme d'un graphe.

Diverses méthodes empiriques permettent d'éviter les minima locaux : la méthodes des directions tabou en est une. D'autre part, certaines approximations des améliorations de la méthode du gradient peuvent être appliquées approximativement dans les espaces discrets.

L'espace des hypothèses comme un graphe

Il est fréquent que l'on puisse définir une structure de graphe sur l'ensemble des hypothèses. Sans rentrer dans une définition formelle, la signification de cette structure est la suivante : chaque hypothèse peut, par des modifications syntaxiques simples, être transformée en un certain nombre d'autres hypothèses qui lui sont proches de ce point de vue. Autrement dit, si l'hypothèse de départ est formalisée comme le noeud d'un graphe, elle est reliée par des arcs à un certain nombre d'autres noeuds de ce graphe. Les arcs peuvent d'ailleurs être valués pour exprimer le coût de la transformation correspondante.

Par exemple, on verra dans le chapitre 7, sur l'inférence grammaticale, qu'une hypothèse est un automate fini et que toutes les autres hypothèses proches sont obtenues par une opération qui consiste à fusionner deux états de cet automate. La figure 3.11 représente cette situation :

17. Mais on ne sait pas le démontrer.

18. Par opposition à la notion d'heuristique qui est seulement une fonction d'estimation de la proximité au but.

l'hypothèse courante est un nœud du graphe (représenté comme le rectangle le plus haut dans la figure) dans lequel est dessiné l'automate correspondant à trois états. Il est relié à trois nœuds (trois rectangles en dessous) par des arcs (des flèches en gras) correspondant aux trois fusions possibles de deux états de l'automate du dessus.

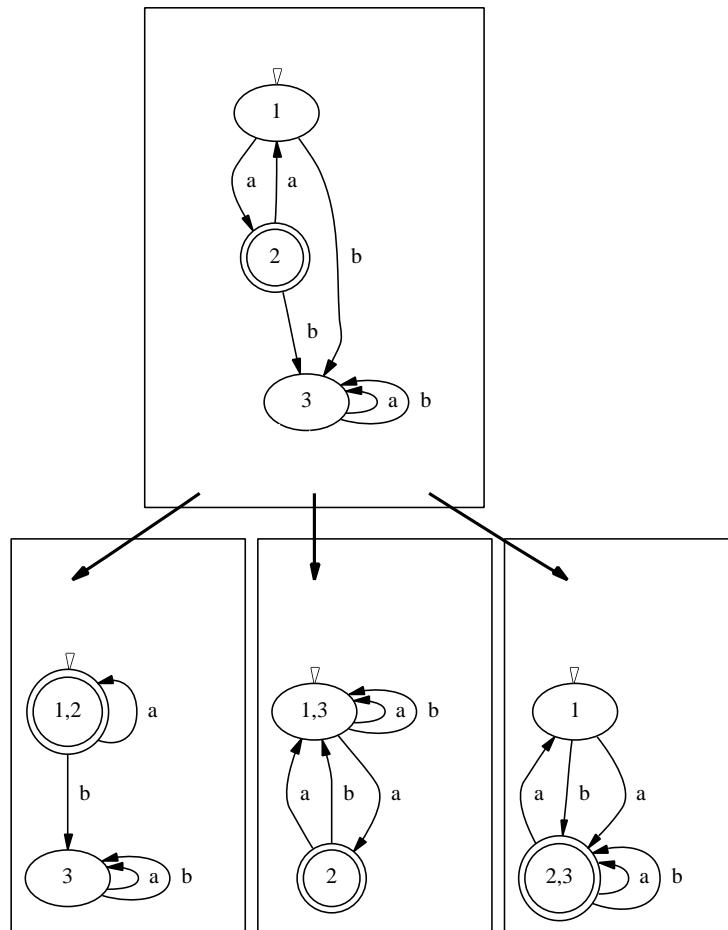


FIG. 3.11 – Une toute petite partie d'un espace d'hypothèses structuré en graphe

Dans cette organisation en graphe, la recherche de la meilleure hypothèse peut être vue comme la construction d'un chemin entre une ou plusieurs hypothèses de départ et celle sur laquelle on décide de s'arrêter. Comme les graphes dans lesquels on travaille sont en général très grands, il est hors de question de les mettre entièrement en mémoire : on procède donc par exploration locale en utilisant des heuristiques pour essayer de trouver une solution aussi bonne que possible, à défaut de la meilleure.

Les techniques employées sont longuement décrites dans nombre de livres d'algorithme et d'intelligence artificielle. Présentons-en une qui est fondée sur une approximation sommaire de la technique du gradient. On la connaît sous le nom d'escalade (*hill climbing*). Dans la version donnée dans l'algorithme 3.1, les arcs sont valués par une valeur strictement positive. La longueur d'un chemin est la somme des longueurs des arcs qui le composent.

Le recuit simulé

L'idée de recuit simulé introduite indépendamment par Kirkpatrick, Gelatt et Vecchi [KGV83]

Algorithme 3.1 L'escalade (*hill-climbing*)

Soit L_0 une liste de noeuds de départ, triés par valeur croissante du risque empirique

$L \leftarrow L_0$

$TEST \leftarrow FAUX$

tant que le test d'arrêt $TEST$ n'est pas *VRAI faire*

 Soit n le premier noeud dans L (si L est vide, ECHEC).

 Enlever n de L

 Mettre dans L tous les noeuds que l'on peut atteindre par un arc à partir de n

 Calculer le risque empirique des hypothèses correspondantes et ranger les noeuds dans L en préservant l'ordre croissant

si le risque empirique du premier noeud de L est supérieur à celui de n **et** ce premier noeud n'est pas dans L_0 **alors**

$TEST \leftarrow VRAI$

fin si

fin tant que

n est l'hypothèse trouvée

et par Černy [Cer85], s'inspire de la métallurgie¹⁹. Plus précisément, si du métal en fusion est refroidi suffisamment lentement, il se solidifie dans une structure d'énergie minimale qui améliore ses caractéristiques. Une analogie avec un problème d'optimisation combinatoire est alors envisageable : aux différents états du système physique correspondent différentes solutions du problème et à l'énergie d'un état correspond la performance d'une solution. On cherche dans les deux cas une configuration d'énergie minimale. La méthode de recuit simulé est une traduction algorithmique de cette analogie. Il s'agit en fait d'une variante stochastique de la technique de gradient dans laquelle au lieu de modifier systématiquement la solution courante en une solution du voisinage meilleure, on accepte de temps en temps, en fonction d'un paramètre de « température », de sélectionner une solution de performance moindre. Cette technique peut permettre d'échapper aux minima locaux, sous certaines conditions portant sur l'évolution de la température, et donc conduire à la solution optimale.

Dans la vaste littérature publiée sur ce sujet, nous ne pouvons que recommander la lecture de [DHS01] pp.351-360 comprenant d'excellentes illustrations du processus et qui aident grandement à sa compréhension.

Les méthodes tabou

Les méthodes tabou ont été introduites par Glover [Glo86, Glo89a, Glo89b]. L'algorithme de base examine la totalité du voisinage et choisit la solution qui améliore le plus la mesure de performance. Lorsqu'il n'y en a pas, la solution sélectionnée est celle qui dégrade le moins la performance. Il est ainsi possible d'échapper aux optima locaux.

Afin d'éviter un retour aux mêmes solutions, une liste circulaire appelée *liste tabou* stocke soit l'ensemble des s derniers mouvements effectués (s est la longueur de la liste tabou), soit des informations représentatives de ces mouvements, soit encore les s dernières solutions rencontrées, ou des informations représentatives de ces solutions. La liste tabou interdit un certain nombre de mouvements, soit parce qu'ils sont tabou soit parce qu'ils conduisent à des solu-

19. D'après le *Petit Robert*, le *recuit* est une « opération thermique destinée à améliorer les qualités mécaniques d'un métal, d'un alliage ».

tions tabou. Comme il arrive cependant que certains mouvements tabou vaillent parfois la peine d'être exécutés, il existe des *critères d'aspiration* permettant d'effectuer un mouvement tabou s'il est jugé intéressant. Le critère d'aspiration le plus courant est le suivant : un mouvement est effectué malgré son caractère tabou s'il permet d'obtenir une solution meilleure que toutes celles obtenues jusque-là.

Les algorithmes par évolution simulée

Ces algorithmes introduits en particulier par Holland [Hol75] s'inspirent des processus de la théorie de l'évolution. Contrairement aux méthodes classiques, ces algorithmes travaillent sur une population de solutions provisoires. À chaque « génération », une nouvelle population est calculée à partir de la précédente grâce à des « opérateurs génétiques » tels que la mutation et le croisement, en favorisant les descendants des meilleurs individus de la génération courante. Dans de nombreux cas, après plusieurs itérations, la population converge vers les régions les plus intéressantes de l'espace de recherche et le meilleur individu obtenu correspond à une solution quasi optimale.

Le chapitre 8 est consacré à ce type d'approche et à des extensions à l'apprentissage symbolique.

Cette section, bien que déjà longue, est très loin d'être exhaustive et laisse donc de côté nombre de techniques d'optimisation intéressantes, par exemple le gradient conjugué pour n'en citer qu'une. Nous renvoyons le lecteur aux livres de référence pour approfondir ces techniques. Par exemple, [DHS01] introduit nombre de méthodes d'optimisation au fur et à mesure des algorithmes d'apprentissage présentés.

3.3.3.3 Les méthodes de Monte Carlo

Lorsqu'un problème de calcul numérique est difficile à résoudre par les méthodes conventionnelles, il est quelquefois possible de lui associer un processus stochastique dont l'une des caractéristiques s'identifie à une valeur numérique recherchée. Une réalisation répétée du processus peut permettre de dégager, par statistique effectuée sur les résultats, une approximation des grandeurs voulues. Une telle démarche est généralement appelée *méthode de Monte Carlo*.

Un exemple classique est l'approximation numérique du nombre irrationnel $1/\pi$ par des jets répétés d'une aiguille sur un réseau de droites équidistantes (de la longueur de l'aiguille). Il s'agit là de la méthode de l'« aiguille de Buffon ».

Plus généralement, ces méthodes peuvent s'appliquer à l'évaluation d'intégrales multidimensionnelles. Elles donnent aussi des informations sur les solutions de systèmes différentiels. Dans le cadre de l'apprentissage artificiel, on les utilise par exemple pour estimer des densités de probabilités, en particulier dans l'approche bayésienne (voir le chapitre 17) et dans l'apprentissage de réseaux bayésiens (voir le chapitre 12).

3.4 L'évaluation de l'apprentissage

Nous disposons d'un ensemble d'exemples et d'un algorithme d'apprentissage dont nous pouvons régler certains paramètres. Cet algorithme nous retourne une hypothèse. Comment évaluer la performance de cette hypothèse ? Suffit-il par exemple de faire confiance au principe *ERM* et de se fonder sur la performance mesurée sur l'échantillon d'apprentissage ? Surtout pas, du moins sans précautions. En effet, non seulement la performance en apprentissage, ce que nous avons appelé *risque empirique* dans le chapitre 2, est intrinsèquement optimiste, mais en outre son comportement n'est pas forcément un bon indicateur de la vraie performance (le *risque réel*). Un phénomène classique, déjà présenté au chapitre 1 est schématisé sur la figure 3.4

dans lequel le risque empirique diminue au fur et à mesure que le système prend en compte davantage d'information (soit par un accroissement du nombre d'exemples présentés, soit par une répétition des exemples d'apprentissage) tandis que le risque réel, d'abord décroissant, se met à augmenter après un certain stade. Ce phénomène est appelée *surapprentissage (over-fitting)*. Un développement théorique et un exemple pratique sont donnés au paragraphe 3.4.4.

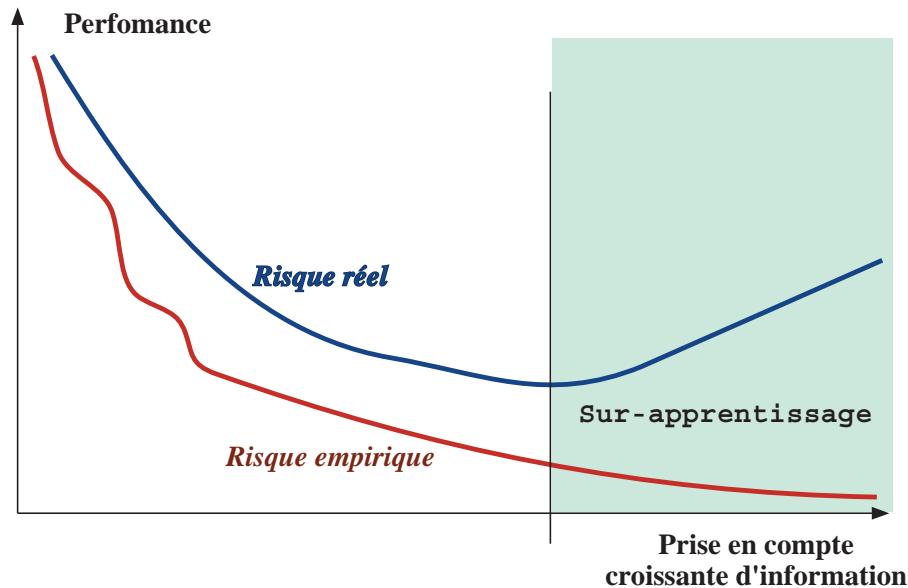


FIG. 3.12 – Illustration du phénomène de surapprentissage. Tandis que le risque empirique continue de diminuer au fur et à mesure de la prise en compte d'information, le risque réel qui diminuait également dans un premier temps, commence à réaugmenter après un certain stade. Il n'y a alors plus de corrélation entre le risque empirique et le risque réel.

Le risque empirique ne peut donc à lui seul servir de base à l'estimation de la performance de l'apprentissage réalisé. Comment doit-on alors procéder ?

3.4.1 L'évaluation *a priori*: critères théoriques

Une solution consiste à appliquer les résultats théoriques décrits dans le chapitre 2 qui fournissent des bornes en probabilité sur le risque réel en fonction du risque empirique. Ces bornes prennent la forme générale :

$$R_{Réel}(h) = R_{Emp}(h) + \Phi(d_{VC}(\mathcal{H}), m)$$

où Φ est une fonction de la dimension de Vapnik-Chervonenkis de l'espace d'hypothèse \mathcal{H} et m est la taille de l'échantillon d'apprentissage \mathcal{S} .

Si on peut obtenir ainsi en théorie des bornes asymptotiquement serrées, les hypothèses qu'il faut faire en pratique pour calculer $\Phi(d_{VC}(\mathcal{H}), m)$ impliquent souvent de telles marges que les bornes obtenues de cette manière sont trop lâches et ne permettent pas d'estimer précisément la performance réelle. C'est pourquoi, sauf cas particuliers favorables, l'estimation de la performance en apprentissage s'opère généralement par des mesures empiriques.

3.4.2 L'évaluation empirique *a posteriori*: généralités

Nous admettons la plupart du temps dans ce paragraphe que l'algorithme d'optimisation employé pour l'apprentissage a parfaitement fonctionné, au sens où il a découvert l'hypothèse la meilleure dans le cadre *ERM*. C'est une simplification irréaliste, mais sans importance pour les développements présentés ici. Pour revenir au cas pratique, il suffit de se souvenir qu'en général le risque empirique de l'hypothèse trouvé par l'algorithme n'est pas minimal.

1. On note $h_{\mathcal{H}}^*$ la règle qui minimise $R_{Réel}(h)$ pour h parcourant \mathcal{H} :

$$h_{\mathcal{H}}^* = \underset{h \in \mathcal{H}}{\text{ArgMin}} R_{Réel}(h)$$

Son risque réel $R_{Réel}(h_{\mathcal{H}}^*)$ peut se noter plus simplement $R_{Réel}(\mathcal{H})$, puisque cette règle ne dépend que de \mathcal{H} .

Cette règle est théoriquement celle que tout algorithme d'apprentissage devrait chercher à approcher. Mais cette quête est illusoire: on ne peut pas en réalité savoir si on en est proche ou pas. L'hypothèse faite par la méthode *ERM* est qu'on peut remplacer sa recherche par celle de la règle $h_{\mathcal{S}, \mathcal{H}}^*$, décrite au paragraphe suivant.

Le risque empirique de $h_{\mathcal{H}}^*$ sur les données d'apprentissage peut se noter $R_{Emp}(h_{\mathcal{H}}^*)$, mais ce terme n'est en général pas mesurable puisque $h_{\mathcal{H}}^*$ est inconnu.

2. On note $h_{\mathcal{S}, \mathcal{H}}^*$ la règle qui minimise le risque empirique sur l'échantillon d'apprentissage \mathcal{S} :

$$h_{\mathcal{S}, \mathcal{H}}^* = \underset{h \in \mathcal{H}}{\text{ArgMin}} R_{Emp}(h)$$

Cette règle est celle que l'algorithme d'apprentissage cherche à trouver en utilisant \mathcal{S} quand il fonctionne selon le principe *ERM*. $h_{\mathcal{S}, \mathcal{H}}^*$ dépend de \mathcal{H} et de \mathcal{S} .

Cette recherche est pertinente: on a vu ci-dessus que dans la plupart des cas, il s'agit d'un problème d'optimisation. Comme on n'est jamais sûr que l'algorithme choisi trouve cette règle, on ne connaît ni son risque empirique $R_{Emp}(h_{\mathcal{S}, \mathcal{H}}^*)$ ni son risque réel $R_{Réel}(h_{\mathcal{S}, \mathcal{H}}^*)$.

3. On note

$$h_{algo, \mathcal{S}, \mathcal{H}}^*$$

la règle trouvée par l'algorithme d'apprentissage. Elle dépend de \mathcal{H} , de \mathcal{S} et de l'algorithme d'apprentissage. Son risque empirique $R_{Emp}(h_{algo, \mathcal{S}, \mathcal{H}}^*)$ se mesure sur l'échantillon d'apprentissage. Son risque réel $R_{Réel}(h_{algo, \mathcal{S}, \mathcal{H}}^*)$ peut s'estimer par des méthodes que nous verrons dans la suite de ce chapitre.

Comme on l'a dit, on suppose pour le moment que $h_{algo, \mathcal{S}, \mathcal{H}}^* = h_{\mathcal{S}, \mathcal{H}}^*$, c'est-à-dire que l'algorithme est efficace de point de vue du principe *ERM*.

Mais en réalité, on a :

$$R_{Emp}(h_{algo, \mathcal{S}, \mathcal{H}}^*) \geq R_{Emp}(h_{\mathcal{S}, \mathcal{H}}^*)$$

et la plupart du temps²⁰:

$$R_{Réel}(h_{algo, \mathcal{S}, \mathcal{H}}^*) \geq R_{Réel}(h_{\mathcal{S}, \mathcal{H}}^*)$$

20. Sauf si on règle l'apprentissage avec un échantillon de validation \mathcal{V} qui est une partie de \mathcal{S} (voir au paragraphe 3.4.5.1).

3.4.3 Risque empirique et risque réel

Soit un ensemble d'apprentissage $\mathcal{S} = \{(\mathbf{x}_i, \mathbf{u}_i)\}$ de taille m et une hypothèse h choisie dans \mathcal{H} pour approcher la fonction cible f . Rappelons la définition générale du chapitre 2.

Définition 3.3 (risque empirique)

Le risque empirique de h est donné par :

$$R_{Emp}(h) = \frac{1}{m} \sum_{i=1}^m l(\mathbf{u}_i, h(\mathbf{x}_i))$$

où $l(\mathbf{u}_i, h(\mathbf{x}_i))$ est par définition la *fonction de perte* qui mesure la différence entre le résultat de l'hypothèse et la valeur de la cible sur un exemple (voir le chapitre 1).

Le risque réel, ou espérance de risque, est défini par :

$$R(h) = \int_{\mathbf{x} \in X, \mathcal{D}} l(h(\mathbf{x}), f(\mathbf{x})) d\mathbf{x} \quad (3.3)$$

l'intégrale est prise sur l'ensemble des formes $\mathbf{x} \in \mathcal{X}$ possibles (voir le chapitre 2).

3.4.3.1 Le cas de la classification

Dans le cas d'un problème de classification, nous prenons la fonction de perte la plus simple :

$$l(\mathbf{u}_i, h(\mathbf{x}_i)) = \begin{cases} 0 & \text{si } \mathbf{u}_i = h(\mathbf{x}_i) \\ 1 & \text{si } \mathbf{u}_i \neq h(\mathbf{x}_i) \end{cases}$$

Avec cette fonction de perte, le risque $R_{Réel}(h)$ mesure la probabilité de mauvaise classification.

Notons m_{err} le nombre d'exemples de l'ensemble d'apprentissage \mathcal{S} qui sont mal classés par une certaine hypothèse h choisie dans \mathcal{H} . Le risque empirique (ou taux d'erreur apparent) de h se mesure alors simplement par :

$$R_{Emp}(h) = \frac{m_{err}}{m}$$

On détaille souvent les résultats ainsi :

Définition 3.4

La matrice de confusion empirique $M_{emp}(i, j)$ d'une règle de classification h est une matrice $C \times C$ dont l'élément générique donne le nombre d'exemples de l'ensemble d'apprentissage \mathcal{S} de la classe i qui ont été classés dans la classe j .

La somme des termes non diagonaux divisée par la taille m de l'ensemble d'apprentissage n'est autre que le risque empirique (ou erreur apparente) de la règle de classification.

3.4.3.2 La qualité d'une hypothèse

Toujours dans le cas de la fonction de perte précédente, le risque réel d'erreur $R_{Réel}(h)$ est la probabilité d'erreur de h , c'est-à-dire la probabilité que h classe mal un objet.

Quand on a choisi une certaine règle h , on sait que quand m augmente, son risque apparent $R_{Emp}(h)$ converge en probabilité vers son risque réel $R_{Réel}(h)$, en vertu de l'application de loi des grands nombres par les théorèmes de Vapnik (voir le chapitre 2).

Mais ceci ne nous indique pas comment choisir \mathcal{H} ni comment y sélectionner h pour approcher $h_{\mathcal{H}}^*$, la règle de \mathcal{H} qui minimise le risque réel.

Une technique naturelle est, comme on l'a dit, d'appliquer le principe *ERM*, donc d'essayer, par un algorithme d'apprentissage, de sélectionner dans \mathcal{H} la règle $h_{\mathcal{S}, \mathcal{H}}^*$ qui minimise le taux d'erreur empirique sur l'ensemble d'apprentissage \mathcal{S} .

Cette approche fournit une règle biaisée de manière optimiste :

$$R_{Emp}(h_{\mathcal{S}, \mathcal{H}}^*) \leq R_{Réel}(h_{\mathcal{S}, \mathcal{H}}^*) \leq R_{Réel}(h_{\mathcal{H}}^*) = R_{Réel}(\mathcal{H})$$

La façon la plus naturelle d'estimer rigoureusement $R_{Réel}(h_{\mathcal{S}, \mathcal{H}}^*)$ est d'utiliser un *échantillon de test* \mathcal{T} statistiquement indépendant de \mathcal{S} pour faire cette mesure, mais il existe d'autres méthodes. On y reviendra au paragraphe 3.4.5.

3.4.4 La sélection de modèle en pratique

Nous cherchons donc à approcher $h_{\mathcal{S}, \mathcal{H}}^*$. Le choix de l'espace des hypothèses que l'on explore étant laissé libre, il serait utile de savoir *a priori* comparer deux espaces \mathcal{H} et \mathcal{H}' . Mais on ne dispose en général d'aucune indication sur le sujet. En revanche, une fois choisi \mathcal{H} , il est souvent facile de l'ordonner partiellement en fonction d'un critère. On peut souvent indexer ses éléments par l'ordre de la complexité algorithmique du programme qui réalise la fonction de décision correspondante et paramétriser le programme d'apprentissage selon cet index.

Par exemple, on verra au chapitre 11 l'apprentissage de règles de classification représentées par des *arbres de décision*. On peut ordonner l'espace \mathcal{H} de ces arbres en caractérisant un élément par le nombre k de ses noeuds. On verra que cette mesure est directement liée au temps moyen de calcul que prend le programme correspondant pour classer une donnée inconnue.

Cette approche a été évoquée sous le nom de *sélection de modèle* au chapitre 2. Nous allons l'examiner de nouveau ici pour en donner un exemple.

On supposera donc que l'on peut définir une suite d'ensembles \mathcal{H}_k de complexité²¹ croissante avec k parcourant N et en relation d'inclusion :

$$\mathcal{H}_1 \subset \dots \subset \mathcal{H}_k \subset \mathcal{H}_{k+1} \subset \dots \subset \mathcal{H}_\infty$$

Nous supposons également que la fonction cible finit par être incluse dans un de ces ensembles de taille croissante :

$$f \in \mathcal{H}_\infty$$

Notons maintenant :

- $h_{\mathcal{S}, \mathcal{H}_k}^*$ la règle ayant le risque réel (la probabilité d'erreur) la plus faible de \mathcal{H}_k .
- $h_{\mathcal{S}, \mathcal{H}_k}^*$ la règle ayant le risque empirique (le taux d'erreur apparent) le plus faible de \mathcal{H}_k .

Rappelons que nous faisons l'hypothèse simplificatrice que l'algorithme d'apprentisage employé est idéal du point de vue *ERM* : il est supposé capable de découvrir pour tout ensemble d'apprentissage la règle $h_{\mathcal{S}, \mathcal{H}_k}^*$ dans \mathcal{H}_k . Autrement dit, que les règles $h_{\mathcal{S}, \mathcal{H}_k}^*$ et $h_{algo, \mathcal{S}, \mathcal{H}_k}^*$ sont identiques pour tout \mathcal{S} . Ce n'est (hélas) pas le cas général, mais cela ne change pas les considérations qui vont suivre. La figure 3.14 montre ce qui se produit en général quand cette hypothèse n'est pas respectée.

Nous supposons aussi que la valeur k représente un critère pertinent de complexité ; autrement dit, une bonne procédure de classification dans \mathcal{H}_k doit être capable de décrire les données d'apprentissage de manière de plus en plus précise au fur et à mesure que k augmente.

21. Ce mot peut être pris dans son sens algorithmique.

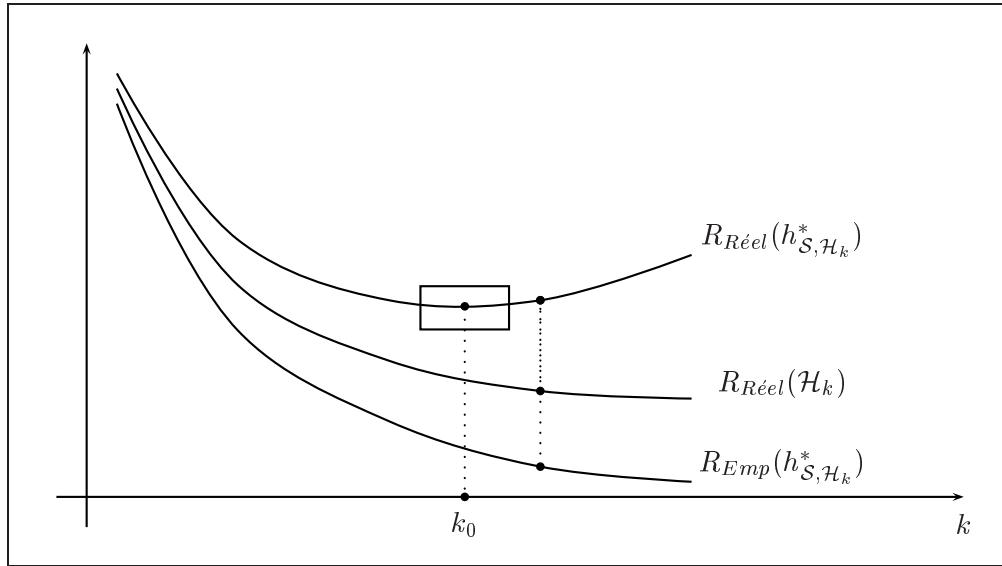


FIG. 3.13 – Le surapprentissage : variation des risques d'erreur quand la complexité k de l'espace des hypothèses \mathcal{H}_k augmente. On remarque les phénomènes suivants :

- (1) Le risque empirique $R_{Emp}(h_{S,H_k}^*)$ de l'hypothèse trouvée par l'algorithme sur l'ensemble d'apprentissage tend vers 0.
- (2) Le risque réel $R_{Reel}(S, \mathcal{H}_k)$ de la meilleure hypothèse de \mathcal{H}_k tend aussi vers 0, mais moins vite. Aucun algorithme ne peut la trouver avec certitude.
- (3) Le risque réel $R_{Reel}(h_{S,H_k}^*)$ de l'hypothèse trouvée par l'algorithme sur l'ensemble d'apprentissage décroît jusqu'à k_0 , puis augmente. Le problème est donc de trouver la région de k_0 , la meilleure compte tenu de l'algorithme employé et des données d'apprentissage.

On a supposé que l'algorithme d'apprentissage était capable de trouver $h_{\mathcal{H}_k}^*$, c'est-à-dire qu'il fonctionne selon le principe ERM et ne commet pas d'erreur d'approximation.

Que peut-on dire des valeurs relatives des taux apparents d'erreur et des probabilités d'erreur de $h_{\mathcal{H}_k}^*$ et de $h_{\mathcal{H}_k}^*$ pour un k donné et quand k varie ?

- **k est constant**

1. On a tout d'abord :

$$R_{Reel}(h_{S,H_k}^*) \geq R_{Reel}(h_{\mathcal{H}_k}^*)$$

Cette inégalité traduit simplement le fait que la règle la règle h_{S,H_k}^* , supposée trouvée par l'algorithme, n'est en général pas optimale en terme d'erreur réelle, parce que l'ensemble d'apprentissage ne peut pas parfaitement résumer les distributions de probabilité des classes. C'est le problème de toute généralisation.

2. On a aussi en général :

$$R_{Reel}(h_{\mathcal{H}_k}^*) \geq R_{Emp}(h_{S,H_k}^*)$$

Cette formule exprime le fait que la règle apprise étant réglée sur les données d'apprentissage, elle a tendance à surestimer leurs caractéristiques au détriment d'une généralisation exacte, dans l'approche ERM.

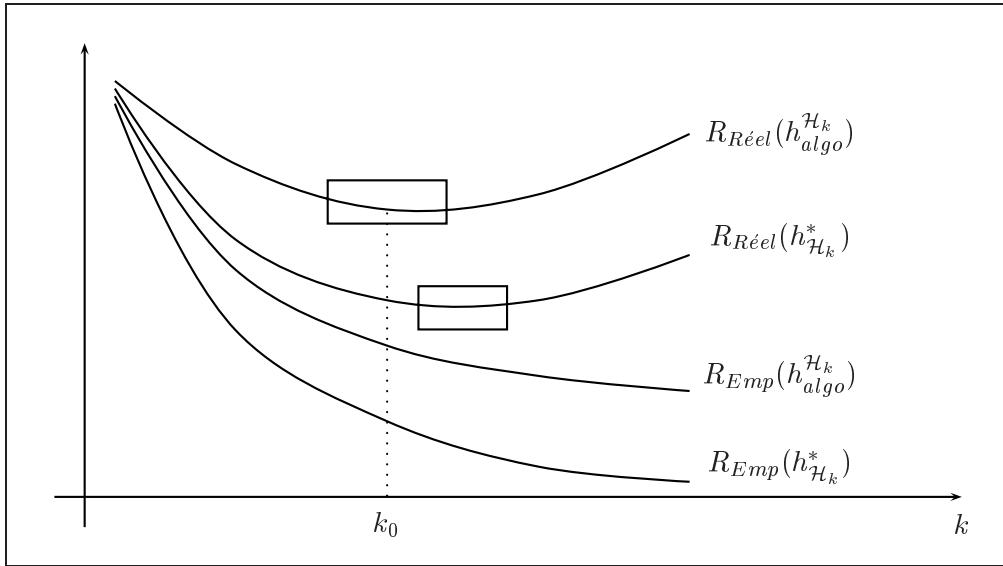


FIG. 3.14 – La cas où l'algorithme d'apprentissage ne trouve pas $h_{H_k}^*$, mais une règle $h_{algo}^{H_k}$.

- **k augmente**

1. On a d'abord en général, pour tout k :

$$R_{Réel}(\mathcal{H}_k) \text{ décroît quand } k \text{ augmente}$$

$$R_{Emp}(h_{\mathcal{S}, \mathcal{H}_k}^*) \text{ décroît quand } k \text{ augmente}$$

En effet, l'erreur apparente de $h_{\mathcal{H}_k}^*$ diminue quand k augmente, en général jusqu'à valoir 0 pour k assez grand : dans un espace de règles assez complexe, on peut faire l'apprentissage par cœur d'un échantillon \mathcal{S} .

En général la valeur :

$$R_{Réel}(h_{\mathcal{S}, \mathcal{H}_k}^*) - R_{Emp}(h_{\mathcal{S}, \mathcal{H}_k}^*)$$

est positive et augmente avec k .

2. On a aussi, quand k augmente, jusqu'à un certain²² k_0 :

$$R_{Réel}(h_{\mathcal{S}, \mathcal{H}_1}^*) \leq R_{Réel}(h_{\mathcal{S}, \mathcal{H}_2}^*) \leq \dots \leq R_{Réel}(h_{\mathcal{S}, \mathcal{H}_{k_0-1}}^*) \leq R_{Réel}(h_{\mathcal{S}, \mathcal{H}_{k_0}}^*)$$

Ce qui signifie qu'augmenter k semble avoir un effet positif, puisque la probabilité d'erreur de la règle apprise tend à diminuer.

3. Mais au dessus de k_0 , l'inégalité s'inverse:

$$k \geq k_0 : R_{Réel}(h_{\mathcal{H}_k}^*) \geq R_{Réel}(h_{\mathcal{H}_{k+1}}^*) \geq \dots$$

On rencontre donc une valeur k_0 au-delà de laquelle compliquer la famille des règles ne sert plus à rien, puisque la performance réelle du classificateur appris diminuera.

22. Pour simplifier, on suppose qu'il existe une valeur unique ; en réalité, il s'agit d'une région autour de cette valeur. Mais cela ne change pas l'argument de fond.

Ce dernier phénomène est appelé le *surapprentissage* (Voir la figure 3.16). Intuitivement, il signifie qu'une règle de classification de trop grande complexité représente trop exactement l'ensemble d'apprentissage, c'est-à-dire réalise en pratique un apprentissage par coeur, au détriment de sa qualité de généralisation. Il existe par conséquent une valeur k_0 de compromis, sur laquelle on ne possède *a priori* aucune information, qui est la meilleure pour un échantillon d'apprentissage donné et une famille de règles de classification ordonnée selon la complexité k . La valeur k_0 est donc critique. Pour l'estimer, on peut utiliser un échantillon de validation \mathcal{V} (voir le paragraphe 3.4.6).

Dans le cas (général en pratique) où l'algorithme d'apprentissage n'est pas optimal du point de vue *ERM*, la figure 3.14 illustre le même phénomène.

Un exemple

La figure 3.15 présente un exemple de réglage de la valeur k_0 , c'est-à-dire de sélection de modèle, à partir de l'exemple artificiel suivant :

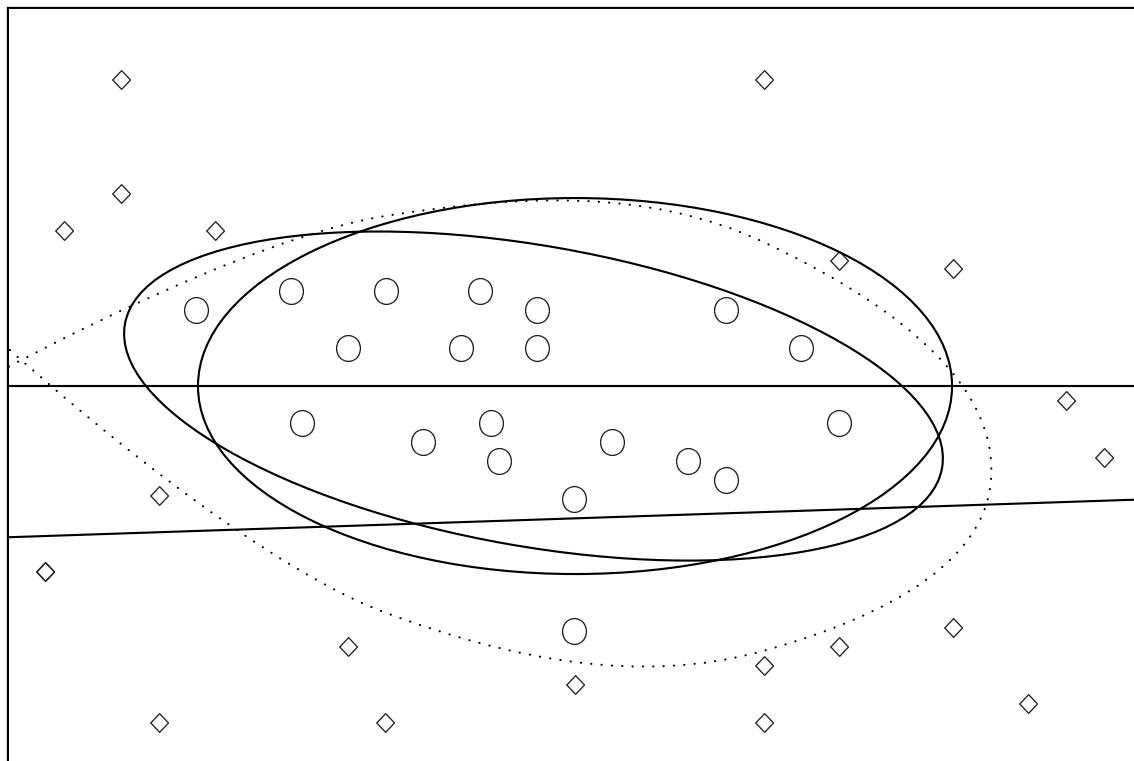


FIG. 3.15 – L'ellipse centrale est la séparatrice bayésienne. On la connaît car les données ont été fabriquées artificiellement. Son erreur réelle vaut : $h_{Bayes} = 5\%$

L'ellipse de biais est l'hypothèse de \mathcal{H}_2 qui minimise l'erreur apparente. Son erreur réelle est légèrement supérieure à h_B .

La droite horizontale est l'hypothèse de \mathcal{H}_1 qui minimise l'erreur réelle.

La droite de biais est l'hypothèse de \mathcal{H}_1 qui minimise l'erreur empirique.

La courbe en pointillés est une hypothèse de \mathcal{H}_4 (une courbe de degré 4) dont l'erreur apparente est nulle mais dont l'erreur réelle est largement supérieure à h_{Bayes} .

Soient deux classes de densité uniforme, l'une à l'intérieur de l'ellipse centrale, l'autre entre

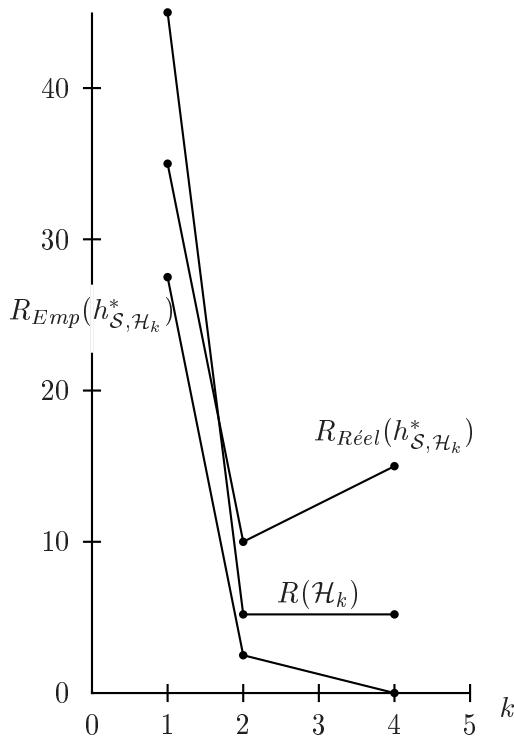


FIG. 3.16 – La sélection de modèle sur un exemple.

le rectangle extérieur et cette ellipse. La séparatrice optimale est ici artificiellement connue : c'est l'ellipse centrale.

Pour rendre le problème un peu plus difficile, nous tirons les coordonnées des points d'apprentissage selon les distributions uniformes, en ajoutant un bruit gaussien. La séparatrice optimale reste l'ellipse, mais les points des deux classes ne sont plus forcément tous exactement de part et d'autre de la séparatrice. Par exemple, les 40 points que l'on voit sur la figure 3.15 constituent un ensemble d'apprentissage : ils ont été tirés indépendamment selon cette distribution uniforme bruitée. On remarque que le bruitage fait que l'un des points \circlearrowleft se trouve à l'extérieur de l'ellipse et un point \diamond à l'intérieur. L'erreur R_{Bayes} n'est donc pas nulle, à cause de ce bruitage. Nous avons fixé cette erreur réelle à 5 %. Sa valeur empirique sur les données d'apprentissage est aussi de 5 % (deux points mal classés sur quarante : $\frac{2}{40} = 5\%$).

Soit \mathcal{H}_1 l'ensemble des droites, \mathcal{H}_2 celui des courbes du second degré, \mathcal{H}_3 celui des courbes du troisième degré, etc.

Pour $k = 1$, la séparatrice optimale $h_{\mathcal{H}_1}^*$ est la droite horizontale au milieu de la figure 3.15.

Pour $k = 2$, on a par hypothèse $h_{\mathcal{H}_2}^* = h_{Bayes}$. La meilleure surface appartient donc à \mathcal{H}_2 . On en est certain ici seulement parce que les données sont artificielles et que h_{Bayes} est connue.

Pour $k \geq 3$, on a : $h_{\mathcal{H}_2}^* = h_{\mathcal{H}_k}^* = h_{Bayes}$, puisqu'on peut ramener une courbe de degré supérieur à une courbe du second degré en annulant les coefficients nécessaires.

La meilleure droite, celle qui minimise l'erreur réelle, se note $h_{\mathcal{H}_1}^*$. Elle est calculable puisque les densités de probabilité des deux classes sont artificielles : c'est la droite horizontale médiane de l'ellipse. Son erreur réelle est aussi calculable : elle vaut $R(\mathcal{H}_1) = 35\%$. Dans notre exemple,

son erreur empirique vaut $\frac{10+7}{20+20} = 42.5\%$ puisque la matrice de confusion empirique est la suivante :

	○	◊
○	10	10
◊	7	13

Par exemple, le chiffre 7 de dans cette matrice signifie que sept objets étiquetés ◊ ont été classés comme ○.

Un algorithme d'apprentissage de bonne qualité trouve dans \mathcal{H}_1 la droite $h_{\mathcal{S}, \mathcal{H}_1}^*$ qui minimise l'erreur empirique. Celle-ci vaut $\frac{10+1}{20+20} = 27.5\%$, puisque sa matrice de confusion empirique est la suivante :

	○	◊
○	19	1
◊	10	10

Comme les distributions sont uniformes et la géométrie fixée, on peut mesurer $R_{Réel}(h_{\mathcal{S}, \mathcal{H}_1}^*)$, qui vaut 45 %.

Dans \mathcal{H}_2 , cet algorithme trouve $h_{\mathcal{S}, \mathcal{H}_2}^*$, pour laquelle on a

$$R_{Emp}(h_{\mathcal{S}, \mathcal{H}_2}^*) = \frac{1+0}{20+20} = 2.5\%$$

C'est la meilleure ellipse que l'on puisse tracer pour séparer les données d'apprentissage : elle ne commet qu'une erreur. $R_{Réel}(h_{\mathcal{S}, \mathcal{H}_2}^*)$ se mesure à environ 10 %.

Nous ne traitons pas le cas de \mathcal{H}_3 et nous passons directement à \mathcal{H}_4 .

Dans \mathcal{H}_4 on peut trouver $h_{\mathcal{S}, \mathcal{H}_4}^*$ telle que $R_{Emp}(h_{\mathcal{S}, \mathcal{H}_4}^*) = 0$, mais cette fois $R_{Réel}(h_{\mathcal{S}, \mathcal{H}_4}^*)$ a augmenté jusqu'à environ 15 % et dépasse $R_{Emp}(h_{\mathcal{S}, \mathcal{H}_2}^*)$.

On a donc $k_0 = 2$ dans cet exemple. En résumé :

Règle	Courbe	Risque empirique	Risque réel
$h_{\mathcal{S}, \mathcal{H}_1}^*$	Droite oblique	27.5 %	45 %
$h_{\mathcal{S}, \mathcal{H}_2}^*$	Ellipse penchée	2.5 %	10 %
$h_{\mathcal{S}, \mathcal{H}_4}^*$	Courbe de degré 4	0 %	15 %
$h_{\mathcal{H}_1}^*$	Droite horizontale	42.5 %	35 %
$h_{\mathcal{H}_2}^* = h_{Bayes}$	Ellipse centrale	5 %	5 %

3.4.5 L'estimation du risque réel d'une hypothèse

3.4.5.1 L'utilisation d'un échantillon de test

Présentation

La méthode la plus simple pour estimer la qualité objective d'une hypothèse d'apprentissage h est de couper l'ensemble des exemples en deux ensembles indépendants : le premier, noté \mathcal{A} , est utilisé pour l'apprentissage de h et le second, noté \mathcal{T} , sert à mesurer sa qualité. Ce second ensemble est appelé *échantillon (ou ensemble d'exemples) de test*. On a $\mathcal{S} = \mathcal{A} \cup \mathcal{T}$ et $\mathcal{A} \cap \mathcal{T} = \emptyset$

Comme nous allons le voir, la mesure des erreurs commises par h sur l'ensemble de test \mathcal{T} est une estimation du risque réel d'erreur de h . Cette estimation se note :

$$\hat{R}_{Réel}(h)$$

Examinons d'abord le cas particulier de l'apprentissage d'une règle de classification.

Le cas de la classification

Rappelons d'abord la définition d'une matrice de confusion :

Définition 3.5

La matrice de confusion $M(i, j)$ d'une règle de classification h est une matrice $C \times C$ dont l'élément générique donne le nombre d'exemples de l'ensemble de test \mathcal{T} de la classe i qui ont été classés dans la classe j .

Dans le cas d'une classification binaire, la matrice de confusion est donc de la forme :

	‘+’	‘-’
‘+’	Vrais positifs	Faux positifs
‘-’	Faux négatifs	Vrais négatifs

Si toutes les erreurs sont considérées comme également graves, la somme des termes non diagonaux de M , divisée par la taille t de l'ensemble de test, est une estimation $\hat{R}_{Réel}(h)$ sur \mathcal{T} du risque réel de h .

$$\hat{R}_{Réel}(h) = \frac{\sum_{i \neq j} M(i, j)}{t}$$

En notant t_{err} le nombre d'objets de l'ensemble de test mal classés, on a donc :

$$\hat{R}_{Réel}(h) = \frac{t_{err}}{t}$$

On a défini plus haut la matrice de confusion empirique : celle de l'ensemble d'apprentissage sur lui-même ; pour cette matrice, la somme des termes non diagonaux est proportionnelle au risque empirique, mais n'est pas une estimation du risque réel.

L'intervalle de confiance de l'estimation

Quelle confiance peut-on accorder à l'estimation $\hat{R}_{Réel}(h)$? Peut-on la traduire numériquement? La réponse à ces deux questions est donnée de manière simple par des considérations statistiques classiques. Si les échantillons aléatoires d'apprentissage et de test sont indépendants alors la précision de l'estimation ne dépend que du nombre t d'exemples de l'ensemble de test et de la valeur de $\hat{R}_{Réel}(h)$.

Il est démontré à l'annexe 18.7 qu'une approximation suffisante dans le cas où t est assez grand (au delà de la centaine) donne l'*intervalle de confiance* de $\hat{R}_{Réel}(h)$ à $x\%$ par la formule :

$$\left[\frac{t_{err}}{t} \pm \zeta(x) \sqrt{\frac{\frac{t_{err}}{t} (1 - \frac{t_{err}}{t})}{t}} \right]$$

La fonction $\zeta(x)$ a en particulier les valeurs suivantes :

x	50 %	68 %	80 %	90 %	95 %	98 %	99 %
$\zeta(x)$	0.67	1.00	1.28	1.64	1.96	2.33	2.58

Par exemple, pour $t = 300$ et $t_{err} = 15$, on a $\hat{R}_{Réel}(h) = 0.2$ et l'intervalle de confiance à 95 % de $\hat{R}_{Réel}(h)$ vaut :

$$\left[0.2 \pm 1.96 \sqrt{\frac{0.2(1 - 0.2)}{300}} \right] \approx [0.25, 0.15]$$

ce qui signifie que la probabilité que $R_{Réel}(h)$ soit à l'intérieur de cet intervalle est supérieure à 95 %.

Si on avait obtenu la même proportion d'erreur sur un échantillon de test de taille 1000, cet intervalle aurait été réduit environ de moitié : [0.225, 0.175].

L'estimation du taux d'erreur réel par une mesure sur un échantillon de test \mathcal{T} indépendant de l'échantillon d'apprentissage \mathcal{A} , fournit une estimation non biaisée de $R_{Réel}(h)$ avec un intervalle de confiance contrôlable, ne dépendant que de la taille t de l'échantillon de test. Plus celle-ci est grande, plus l'intervalle de confiance est réduit et par conséquent plus le taux d'erreur empirique donne une indication du taux d'erreur réel.

Malheureusement, dans la plupart des applications, le nombre d'exemples, c'est-à-dire d'observations pour lesquelles un expert a fourni une étiquette, est limité. Le plus souvent chaque nouvel exemple est coûteux à obtenir et il ne peut donc être question d'augmenter à volonté l'échantillon d'apprentissage et l'échantillon de test. En fait, il existe un conflit entre l'intérêt d'avoir le plus grand échantillon possible \mathcal{A} pour apprendre, et le plus grand échantillon possible \mathcal{T} pour tester le résultat de l'apprentissage. Comme il est nécessaire que les deux échantillons soient indépendants l'un de l'autre, ce qui est donné à l'un est retiré à l'autre. C'est pourquoi cette méthode de validation est appelée *hold-out method* par les anglophones. Elle est envisageable lorsque les données sont abondantes. Si en revanche les données sont parcimonieuses, il faut avoir recours à d'autres méthodes.

3.4.5.2 L'estimation par validation croisée

L'idée de la validation croisée (*N-fold cross-validation*) consiste à :

- Diviser les données d'apprentissage \mathcal{S} en N sous-échantillons de tailles égales.
- Retenir l'un de ces échantillons, disons de numéro i , pour le test et apprendre sur les $N - 1$ autres.
- Mesurer le taux d'erreur empirique $\hat{R}_{Réel}^i(h)$ sur l'échantillon i .
- Recommencer n fois en faisant varier l'échantillon i de 1 à N .

L'erreur estimée finale est donnée par la moyenne des erreurs mesurées :

$$\hat{R}_{Réel}(h) = \frac{1}{N} \sum_{i=1}^N \hat{R}_{Réel}^i(h)$$

On peut montrer que cette procédure fournit une estimation non biaisée du taux d'erreur réel. Il est courant de prendre pour N des valeurs comprises entre 5 et 10. De cette manière, on peut utiliser une grande partie des exemples pour l'apprentissage tout en obtenant une mesure précise du taux d'erreur réel. En contrepartie, il faut réaliser la procédure d'apprentissage N fois.

La question se pose cependant de savoir quelle hypothèse apprise on doit finalement utiliser. Il est en effet probable que chaque hypothèse apprise dépende de l'échantillon i utilisé pour l'apprentissage et que l'on obtienne donc N hypothèses différentes.

Notons d'emblée que si les hypothèses apprises sont très différentes les unes des autres (en supposant que l'on puisse mesurer cette différence), c'est qu'il faut peut-être y voir une indication

de l'inadéquation de l'espace des hypothèses \mathcal{H} . Cela semble en effet montrer une grande variance (en général associée à une grande dimension de Vapnik-Chervonenkis), et donc le risque d'un apprentissage sans valeur. (Voir la section 2.2.1 dans le chapitre 2).

Le mieux est alors de refaire un apprentissage sur l'ensemble total \mathcal{S} . La précision sera bonne et l'estimation du taux d'erreur est connue par les N apprentissage faits précédemment.

3.4.5.3 L'estimation par la méthode du *leave-one-out*

Lorsque les données disponibles sont très peu nombreuses, il est possible de pousser à l'extrême la méthode de validation croisée en prenant pour N le nombre total d'exemples disponibles. Dans ce cas, on ne retient à chaque fois qu'un seul exemple pour le test, et on répète l'apprentissage N fois pour tous les autres exemples d'apprentissage.

Il faut noter que si l'un des intérêts de la validation par *leave-one-out* est de produire des hypothèses moins variables, on montre en revanche que l'estimation du taux d'erreur est souvent plus variable que pour des valiations croisées avec un N plus petit.

Cette méthode présente l'avantage de la simplicité et de la rapidité; cependant, dans les cas où le nombre total d'exemples dont on dispose est restreint, il peut être intéressant de ne plus distinguer les ensembles d'apprentissage et de test, mais d'utiliser des techniques nécessitant plusieurs passes d'apprentissage: on perdra en temps de calcul mais on gagnera en finesse d'estimation relativement à la quantité de données disponibles.

3.4.5.4 Quelques variantes de la méthode de validation croisée : *bootstrap*, *jackknife*

Ces techniques diffèrent des précédentes en ce qu'elle utilise des tirages avec remise dans l'ensemble des exemples. Le procédé est le suivant: on tire aléatoirement un exemple, pour le placer dans un ensemble appelé *bootstrap*²³. Le procédé est répété n fois et l'apprentissage est alors effectué sur l'ensemble bootstrap. Un test est fait sur les exemples non présents dans cet ensemble, donnant une première valeur P_1 des erreurs du classificateur. Un autre test est fait sur l'ensemble complet des exemples, donnant la valeur P_2 . L'ensemble de l'opération est répété K fois. Une certaine combinaison linéaire de la moyenne $\overline{P_1}$ des valeurs P_1 et de la moyenne $\overline{P_2}$ des valeurs P_2 obtenues donne la valeur $\widehat{R}_{R\acute{e}el}(h)$. La théorie ([HTF01]) propose la formule:

$$\widehat{R}_{R\acute{e}el}(h) = 0.636 \times \overline{P_1} + 0.368 \times \overline{P_2}$$

en se basant sur le fait que l'espérance de la proportion des éléments non répétés dans l'ensemble de test est égale à 0.368. Pour de petits échantillons, la méthode bootstrap fournit une estimation remarquablement précise de $R_{R\acute{e}el}(h)$. En contrepartie, elle demande une grande valeur de K (plusieurs centaines), c'est à dire un nombre élevé d'apprentissages de la règle de classification.

Il existe enfin une autre méthode proche mais plus complexe d'estimation appelée *jackknife*²⁴ qui vise à réduire le biais du taux d'erreur en resubstitution, lorsque des données sont utilisées à la fois pour l'apprentissage et pour le test. Nous renvoyons le lecteur intéressé à par exemple [Rip96] pp.72-73 ou à [Web99]. Ce sont également de bonnes références pour le problème de l'estimation de performance en général.

23. On sait que le Baron de Munchausen savait s'élever dans les airs en tirant sur ses bottes. La méthode du même nom donne des résultats tout aussi étonnantes (quoique ici justifiés théoriquement et pratiquement).

24. Ou « couteau suisse »: un outil multifonction bien pratique.

3.4.6 Le réglage des algorithmes par un ensemble de validation

Lorsque l'on cherche à résoudre un problème d'apprentissage, on cherche à se décider pour la meilleure méthode, ce qui implique :

- le choix du principe inductif;
- le choix d'une mesure de performance, ce qui implique souvent celui d'une fonction de coût ;
- le choix d'un algorithme d'apprentissage ;
- le choix de l'espace d'hypothèses, qui dépend en partie du choix de l'algorithme ;
- le réglage de tous les paramètres contrôlant le fonctionnement de l'algorithme.

Généralement l'opérateur essaie plusieurs méthodes sur le problème d'apprentissage afin de déterminer celle qui semble la plus appropriée à la classe de problèmes concernée. Comment doit-il procéder ?

Il faut se méfier d'une approche qui semble naturelle. On pourrait en effet croire qu'il suffit de mesurer pour chaque méthode la performance empirique mesurée à l'aide de l'une des techniques décrites plus haut, dans la section 3.4.5, faisant appel d'une manière ou d'une autre à un échantillon de test. Cela serait cependant commettre la faute de faire dépendre cette mesure du réglage de la méthode alors qu'elle doit en être indépendante. En procédant de la sorte, on s'arrange pour minimiser le risque mesuré sur l'échantillon de test et l'on règle la méthode en fonction de cet échantillon. À force de vouloir minimiser ce risque, on risque d'adapter étroitement la méthode d'apprentissage à cet échantillon de test. Cela est dangereux car il se peut que, comme dans le cas du phénomène de surapprentissage, à tant poursuivre ce but, on s'éloigne d'une diminution du risque réel. C'est pourquoi on prévoit, à côté de l'échantillon d'apprentissage et de l'échantillon de test, un troisième échantillon indépendant des deux autres : l'*échantillon de validation*²⁵, sur lequel on évalue la performance réelle de la méthode.

La technique de séparation de l'ensemble d'apprentissage de celui de test peut être raffinée en mettant en œuvre la sélection de modèle de la manière suivante. On divise les données \mathcal{S} supervisées en trois parties : l'ensemble d'apprentissage \mathcal{A} , l'ensemble de test \mathcal{T} et l'*ensemble de validation* \mathcal{V} .

La séparation des données supervisées en trois ensembles est aussi utile pour déterminer à quel moment convergent certains algorithmes d'apprentissage. On y reviendra en particulier au chapitre 10, à propos des réseaux connexionnistes.

3.4.6.1 Estimation de risque : la courbe *ROC*

Jusqu'ici nous avons essentiellement décrit des méthodes d'évaluation des performances ne prenant en compte qu'un nombre : l'estimation du risque réel. Cependant, dans un contexte de prise de décision, il peut-être utile d'être plus fin dans l'évaluation des performances et de prendre en compte non seulement un taux d'erreur, mais aussi les taux de « faux positifs » et de « faux négatifs » (disponibles à partir de la matrice de confusion, voir au paragraphe 3.4.5.1). Souvent, en effet, le coût de mauvaise classification n'est pas symétrique et l'on peut préférer avoir un taux d'erreur un peu moins bon si cela permet de réduire le type d'erreur le plus coûteux (par exemple il vaut mieux opérer à tort de l'appendice (faux positif), plutôt que de ne pas détecter une appendicite (faux négatif)). La courbe *ROC* (de l'anglais *Receiver Operating Characteristic*)

25. La terminologie est ici mal fixée dans la communauté : certains auteurs définissent « échantillon de test » et « échantillon de validation » à l'inverse. Il faut donc se méfier et vérifier que l'on utilise bien la même convention que son interlocuteur.

permet de régler ce compromis²⁶. Nous en détaillons la construction et l'utilisation dans ce qui suit.

Supposons que l'on caractérise les formes d'entrée (par exemple les patients) par une grandeur qui peut résulter de la combinaison d'examens (par exemple prenant en compte l'âge du patient, les antécédents familiaux, sa pression artérielle, etc.). On peut alors établir un graphique pour chaque classe donnant la probabilité d'appartenir à cette classe en fonction de la grandeur (voir la figure 3.17).

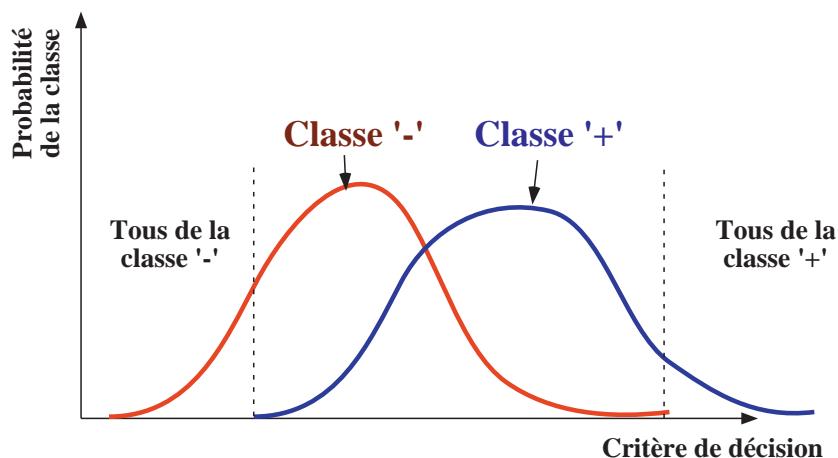


FIG. 3.17 – *Courbes correspondant aux classes ‘+’ et ‘-’.*

Dans une deuxième étape, on détermine pour chaque seuil de la grandeur la probabilité qu'un diagnostic positif soit correct. Cette probabilité est déterminée à partir de la fraction de l'échantillon d'apprentissage pour laquelle la prédiction est exacte. L'aire totale sous la courbe représente cent pour cent de chaque population (voir la figure 3.18).

La troisième étape correspond à la construction de la courbe ROC. Pour chaque seuil, on reporte la proportion de « vrais positifs » en fonction de celle des « faux positifs ». Si l'on obtient une droite, on doit conclure que le test a 50 % de chances de conduire au bon diagnostic. Plus la courbe s'incurve vers le haut, plus le test est pertinent (le rapport des « vrais positifs » sur les « faux positifs » augmente). La pertinence est mesurée par l'aire sous la courbe ; elle augmente avec sa courbure (voir la figure 3.19).

On peut ainsi établir des courbes *ROC* pour plusieurs combinaisons pour n'importe quel système de classification.

Lorsque l'on a trouvé un système de classification jugé suffisamment bon, il reste à choisir le seuil pour un diagnostic ‘classe +’ / ‘classe -’. Le choix du seuil doit fournir une proportion de vrais positifs élevée sans entraîner une proportion inacceptable de faux positifs. Chaque point de la courbe représente un seuil particulier, allant du plus sévère : limitant le nombre de faux positifs au prix de nombreux exemples de la classe ‘+’ non diagnostiqués (forte proportion de faux négatifs, c'est-à-dire faible proportion de vrais positifs), aux plus laxistes : augmentant le nombre de vrais positifs au prix de nombreux faux positifs (voir la figure 3.19). Le seuil optimal pour une application donnée dépend de facteurs tels que les coûts relatifs des faux positifs et faux négatifs, comme de celui de la prévalence de la classe ‘+’. Par exemple un opérateur (de

26. Ces courbes ont été utilisées pour la première fois lors de la deuxième guerre mondiale quand, dans l'usage des radars, on a voulu quantifier leur capacité à distinguer des interférences de nature aléatoire du signal indiquant réellement la présence d'aéronefs.

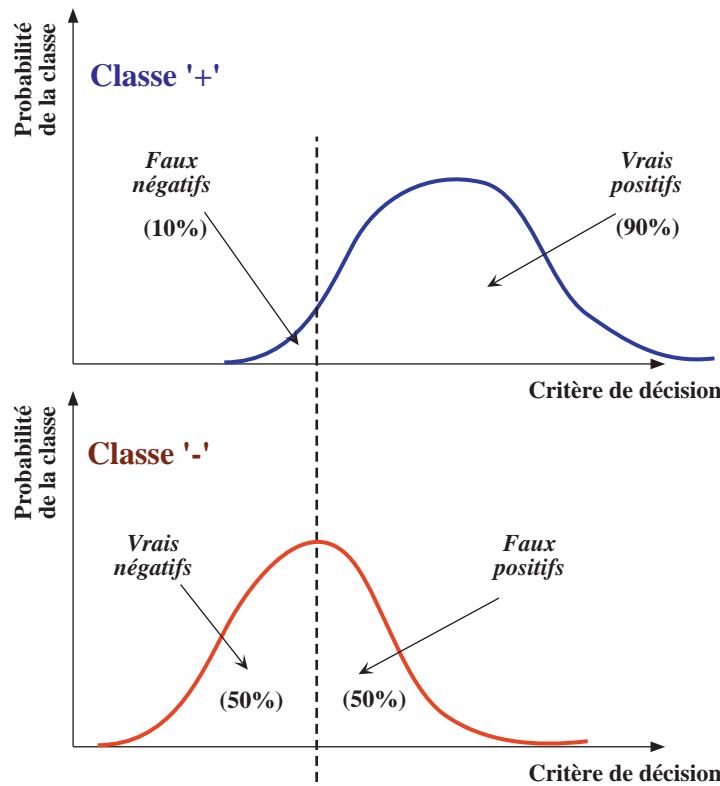


FIG. 3.18 – Seuil décidant pour chaque classe des « vrais positifs », « faux négatifs », « faux positifs » et « vrais négatifs ».

téléphonie ou de chaîne cablée) cherche à détecter des problèmes d'attrition (voir l'avant-propos, page xxi) : quels sont les abonnés susceptibles de le quitter?²⁷. Ces abonnés fuyants sont peu nombreux, mais très coûteux. On cherchera donc à essayer d'en détecter le maximum afin de tenter de les retenir, quitte à détecter aussi quelques faux. On utilisera alors un seuil « laxiste ».

3.4.7 D'autres critères d'appréciation

En plus des critères numériques, il existe un certain nombre de qualités qui permettent de distinguer une hypothèse parmi d'autres.

L'intelligibilité des résultats d'apprentissage

Dans le cas où l'hypothèse apprise est une grammaire ou un ensemble de règles logiques, par exemple, elle peut être munie d'une sémantique directement interprétable dans le langage de l'ingénieur ou de l'expert. Il est important dans ce cas qu'elle soit compréhensible. Cette faculté d'intelligibilité a déjà été évoquée dans l'avant-propos de ce livre (page ix) : nous y avons fait remarquer que la discipline de l'extraction de connaissances dans les données faisait grand cas de cette intelligibilité et que parfois l'apprentissage d'un petit nombre de règles compréhensibles valait mieux qu'un fouillis de règles sophistiquées, même avec une performance objective supérieure.

La simplicité des hypothèses produites

27. On les appelle les *churners* (de *churn* : baratte) dans le jargon du métier. On parle aussi d'attrition (voir l'avant-propos).

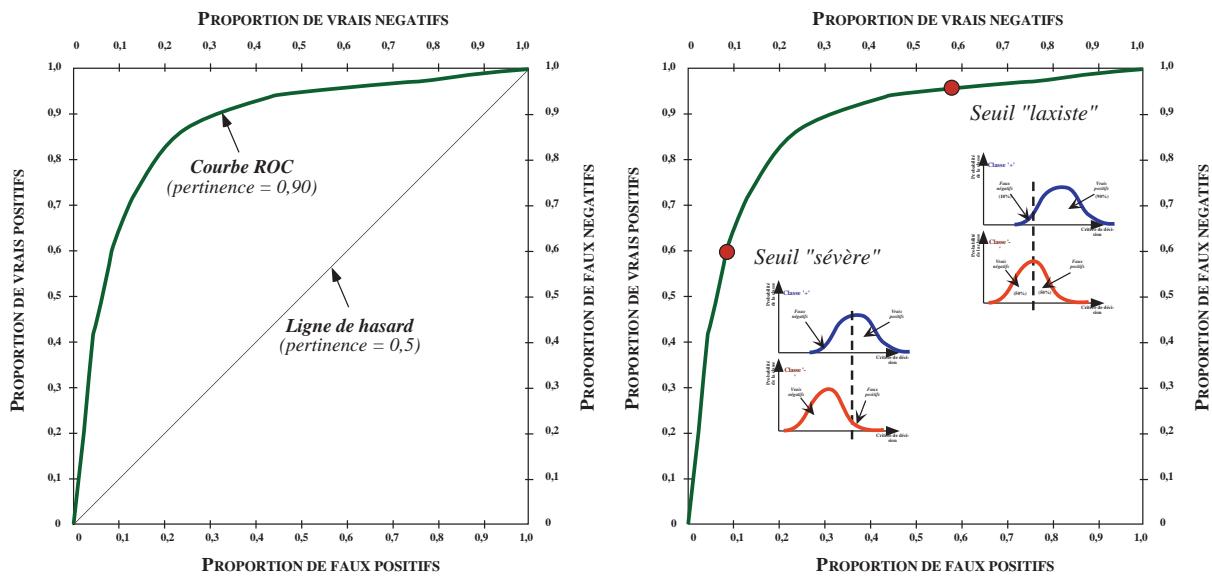


FIG. 3.19 – Une courbe ROC à gauche. Deux seuils sur cette courbe à droite.

Ce critère est relié au précédent. Il relève d'un argument rhétorique classique, le *rasoir d'Occam*, qui affirme qu'il ne sert à rien de multiplier les « entités » inutiles²⁸, autrement dit qu'une explication simple vaut mieux qu'une explication compliquée. Ce principe a été rationalisé par certains chercheurs dans le domaine de la théorie de l'apprentissage [LV97].

Le taux de couverture

Dans le cas d'apprentissage de concept, sans contre-exemples, on utilise parfois la mesure de qualité d'une hypothèse appelée le *taux de couverture*. C'est la proportion d'exemples positifs qui sont couverts par l'hypothèse.

3.5 La comparaison des méthodes d'apprentissage

On peut toujours utiliser différents algorithmes d'apprentissage sur une même tâche. Comment interpréter la différence de performance mesurée empiriquement entre deux algorithmes ? Plus concrètement, est-ce qu'un algorithme dont la performance en taux d'erreur de classification binaire vaut 0.17 est meilleur qu'un autre dont la performance mesurée est de 0.20 ?

La réponse n'est pas évidente, car la performance mesurée dépend à la fois des caractéristiques des tests empiriques effectués et des échantillons de tests utilisés. Départager deux systèmes sur une seule mesure est donc problématique. La même question se pose d'ailleurs pour deux hypothèses produites par le même algorithme à partir de conditions initiales différentes.

Toute une littérature porte sur ce problème et nous n'en esquissons que les grandes lignes dans ce qui suit. On ne saurait être trop attentif à ces questions lorsque l'on teste effectivement des systèmes, sous peine d'affirmer à peu près n'importe quoi.

Lors de la comparaison entre deux algorithmes, ou de deux hypothèses produites par le même

28. *Frustra fit per plura, quod fieri potest per pauciora*, classiquement traduit par : It is vain to do with more what can be done with less. Ou *Essentia non sunt multiplicanda praeter necessitatem*. Entities should not be multiplied unnecessarily. Guillaume d'Occam (1288-1348).

algorithme à partir de conditions initiales différentes, il faut distinguer le cas où l'on se base sur des échantillons de test identiques ou différents. Nous allons traiter rapidement les deux cas.

3.5.1 La comparaison de deux hypothèses produites par un même algorithme sur deux échantillons de test différents.

Soient deux hypothèses h_1 et h_2 produites par le même algorithme d'apprentissage²⁹ et deux ensembles de test \mathcal{T}_1 et \mathcal{T}_2 de taille t_1 et t_2 . On suppose ici que ces deux échantillons de test sont indépendants : ils sont obtenus l'un et l'autre par tirage *i.i.d.* selon la même distribution. Nous cherchons ici à estimer la quantité :

$$\delta_R(h_1, h_2) = R_{R\acute{e}el}(h_1) - R_{R\acute{e}el}(h_2)$$

Si nous disposons d'estimateurs de ces deux risques réels, il est possible de montrer qu'un estimateur de leur différence s'écrit comme la différence des estimateurs :

$$\widehat{\delta}_R(h_1, h_2) = \widehat{R}_{R\acute{e}el}(h_1) - \widehat{R}_{R\acute{e}el}(h_2)$$

En notant $t_{err,1}$ le nombre d'objets de l'ensemble de test \mathcal{T}_1 mal classés par l'hypothèse h_1 et $t_{err,2}$ le nombre d'objets de l'ensemble de test \mathcal{T}_2 mal classés par l'hypothèse h_2 , on a donc :

$$\widehat{\delta}_R(h_1, h_2) = \frac{t_{err,1}}{t_1} - \frac{t_{err,2}}{t_2}$$

L'intervalle de confiance à x % de cette valeur est donné par la formule 3.4 :

$$\left[\widehat{R}_{R\acute{e}el}(h) \pm \zeta(x) \sqrt{\frac{\frac{t_{err,1}}{t_1} \left(1 - \frac{t_{err,1}}{t_1}\right)}{t_1} + \frac{\frac{t_{err,2}}{t_2} \left(1 - \frac{t_{err,2}}{t_2}\right)}{t_2}} \right] \quad (3.4)$$

3.5.2 La comparaison de deux algorithmes sur des ensembles de test différents

Prenons maintenant une situation que l'on rencontre souvent dans la pratique : on dispose de données d'apprentissage et on cherche quel est l'algorithme à leur appliquer, parmi une panoplie disponible. Par exemple si on cherche un concept dans \mathbb{R}^d , donc si les données d'apprentissage sont numériques et leur supervision positive ou négative, on peut chercher une fonction séparatrice sous la forme d'un hyperplan (chapitre 9), ou essayer un réseau connexionniste (chapitre 10), ou prendre la règle des k -ppv (chapitre 14), etc. Comment choisir en pratique la meilleure ?

Supposons avoir à notre disposition deux algorithmes A^1 et A^2 et un ensemble de données supervisées. La méthode la plus simple est de diviser cet ensemble en deux parties disjointes \mathcal{S} et \mathcal{T} et de faire fonctionner A^1 et A^2 sur \mathcal{S} (éventuellement en réglant des paramètres avec une partie \mathcal{V} de \mathcal{S}), puis de comparer les performances sur \mathcal{T} . Deux questions se posent :

- Peut-on accorder une bonne confiance à cette comparaison ?
- Peut-on faire une comparaison plus précise avec les mêmes données ?

La réponse à la première question est plutôt négative. La principale raison en est que l'échantillon d'apprentissage étant le même pour les deux méthodes, ses particularités seront amplifiées par la comparaison. Ce que l'on cherche est quel est le meilleur algorithme non pas sur \mathcal{S} , mais sur la fonction cible dont les exemples de \mathcal{S} ne sont que des tirages aléatoires.

29. Sur deux ensembles d'apprentissage différents ou sur le même, en changeant dans ce cas les conditions initiales, par exemple.

Pour répondre positivement à la seconde question, il faut donc utiliser une technique qui brasse aléatoirement les données d'apprentissage et de test, comme la validation croisée.

Un algorithme efficace est donné ci-dessous (3.2).

Algorithme 3.2 La comparaison de deux algorithmes d'apprentissage

Partitionner les données d'apprentissage $\mathcal{S} \cup \mathcal{T}$ en Q parties égales.

On les note : $\mathcal{T}_1, \mathcal{T}_i, \dots, \mathcal{T}_Q$.

pour $i = 1, Q$ **faire**

$\mathcal{S}_i \leftarrow \mathcal{D} - \mathcal{T}_i$

Faire tourner l'algorithme A^1 sur \mathcal{S}_i . Il fournit l'hypothèse h_i^1 .

Faire tourner l'algorithme A^2 sur \mathcal{S}_i . Il fournit l'hypothèse h_i^2 .

$\delta_i \leftarrow R_i^1 - R_i^2$, où R_i^1 et R_i^2 sont les taux d'erreur de h_i^1 et h_i^2 sur \mathcal{T}_i

fin pour

$\bar{\delta} \leftarrow \frac{1}{Q} \sum_{i=1}^Q \delta_i$

L'intervalle de confiance à x % de $\bar{\delta}$, qui est un estimateur de la différence de performance entre les deux algorithmes, est alors donné par la formule :

$$\left[\bar{\delta} \pm \zeta(x, k) \sqrt{\frac{1}{k(k-1)} \sum_{i=1}^k (\delta_i - \bar{\delta})^2} \right]$$

La fonction $\zeta(x, k)$ tend vers $\zeta(x)$ quand k augmente, mais la formule ci-dessus n'est valable que si la taille de chaque \mathcal{T}_i est d'au moins trente exemples. Donnons-en quelques valeurs :

x	90 %	95 %	98 %	99 %
$\zeta(x, 2)$	2.92	4.30	6.96	9.92
$\zeta(x, 5)$	2.02	2.57	3.36	4.03
$\zeta(x, 10)$	1.81	2.23	2.76	3.17
$\zeta(x, 30)$	1.70	2.04	2.46	2.75
$\zeta(x, +\infty) = \zeta(x)$	1.64	1.96	2.33	2.58

3.5.3 La comparaison de deux algorithmes sur le même ensemble de test

Si les ensembles tests sur lesquels sont évalués les deux algorithmes sont les mêmes, alors les intervalles de confiance peuvent être beaucoup plus serrés dans la mesure où l'on élimine la variance due à la différence entre les échantillons de test.

Dietterich a publié en 1997 un long article ([Die97]) sur la comparaison des algorithmes d'apprentissage sur le même ensemble de test. Il a examiné cinq tests statistiques afin d'étudier la probabilité de détecter une différence entre deux algorithmes alors qu'il n'y en a pas.

Soient donc deux hypothèses de classification h_1 et h_2 . Notons :

- n_{00} = Nombre d'exemples de tests mal classés par h_1 et h_2
- n_{01} = Nombre d'exemples de tests mal classés par h_1 , mais pas par h_2
- n_{10} = Nombre d'exemples de tests mal classés par h_2 , mais pas par h_1
- n_{11} = Nombre d'exemples de tests correctement classés par h_1 et h_2 .

Soit alors la statistique z :

$$z = \frac{|n_{01} - n_{10}|}{\sqrt{n_{10} + n_{01}}}$$

L'hypothèse que h_1 et h_2 aient le même taux d'erreur peut être rejetée avec une probabilité supérieure à $x \%$ si $|z| > \zeta(x)$. La fonction $\zeta(x)$ est la même que celle du paragraphe 3.4.5.1. Ce test est connu sous le nom de test couplé (*paired test*) de McNemar ou de Gillick.

Notes historiques et bibliographiques

La théorie de l'optimisation est un domaine mûr : bien développé et stable. On peut en trouver des exposés synthétiques dans de nombreux ouvrages, par exemple [BSS92, Fle88, Lue84, Man94].

Résumé

Ce chapitre a abordé la représentation des connaissances pour les objets et les concepts, les méthodes de base pour la recherche d'une hypothèse d'apprentissage et les techniques de la mesure de qualité d'une hypothèse. Les points suivants ont été traités :

- Les objets d'apprentissage sont décrits par des attributs qui peuvent être numériques, symboliques, séquentielles, mixtes.
- Avant toute phase d'apprentissage, on peut chercher à nettoyer les données, en particulier en éliminant les attributs inutiles
- Les hypothèses d'apprentissage doivent aussi être représentées. Un grand nombre de possibilités existent : classes, valeurs numériques, arbres, modèles bayésiens et markoviens, grammaires, formules logiques. Toutes ces représentations ne sont pas compatibles avec tous les types de données.
- Une fois l'espace des hypothèses défini, il faut pouvoir l'explorer. Les méthodes de gradient, discret et numérique sont des techniques de base utilisées dans beaucoup d'algorithmes d'apprentissage pour la recherche d'une bonne solution.
- Il faut aussi savoir mesurer la qualité d'une hypothèse produite par un algorithme : les techniques par ensemble de test et ensemble de validation ainsi que les méthodes par validation croisée sont à la base de cette mesure.
- Il est aussi essentiel de savoir comparer deux algorithmes d'apprentissage sur des jeux de données identiques ou non, pour savoir lequel est le plus efficace dans cette situation particulière.

Deuxième partie

Apprentissage par exploration

Chapitre 4

Induction et relation d'ordre : l'espace des versions

La première partie de ce livre a été dévolue aux concepts fondamentaux de l'apprentissage artificiel. Ce chapitre est le premier de ceux destinés à présenter des techniques d'apprentissage. Il présente une technique générale d'induction supervisée de concepts à partir d'exemples et de contre-exemples.

L'apprentissage de concept a longtemps été envisagé uniquement comme une recherche par approximation successive d'une hypothèse cohérente avec l'échantillon de données. La notion d'espace des versions a profondément renouvelé cette manière de voir. De fait, l'espace des versions n'est rien d'autre que l'ensemble de toutes les hypothèses, prises dans un ensemble \mathcal{H} donné a priori, qui sont cohérentes avec les données d'apprentissage. Cet ensemble est en général trop grand pour être manipulé tel quel, mais il se trouve que si une relation de généralité entre hypothèses peut être définie sur \mathcal{H} , alors il devient possible de définir deux ensembles finis S et G qui ont la propriété de définir implicitement tous les concepts cohérents, c'est-à-dire l'espace des versions. L'apprentissage revient alors à construire et à adapter ces deux ensembles frontière : c'est ce que fait l'algorithme d'élimination des candidats.

Ce chapitre constitue un pivot entre les chapitres conceptuels et les chapitres plus appliqués. Il nous permettra d'examiner un exemple clair d'articulation entre la notion d'espace des hypothèses cohérentes discutées dans le chapitre 2 et les principes des méthodes d'exploration de l'espace des hypothèses exposés dans le chapitre 3.

VOICI UN PETIT problème d'apprentissage. J'ai devant moi quatre oiseaux, deux canards et deux manchots. Les attributs suivants sont suffisants pour les décrire : la forme de leur bec, leur *Taille*, leur *Envergure* et la *Couleur* de leur cou. Le premier nous indique si le bec est *Aplati* ou non, les deux suivants se mesurent en centimètres et le dernier peut prendre les valeurs *Roux*, *Orange*, *Gris* ou *Noir*. Ces oiseaux sont étiquetés soit + (les canards), soit - (les manchots) et je veux apprendre un concept cohérent avec les exemples, une formule qui explique tous les canards et rejette tous les manchots.

Je me donne un langage de représentation pour les concepts : je vais les écrire comme une conjonction de certaines propriétés sur les attributs. Par exemple, la formule logique suivante :

$$[Aplati = VRAI] \wedge Taille \in [30, 40] \wedge Envergure \in [-\infty, +\infty] \wedge [Couleur = CouleurChaude]$$

est un concept qui représente l'ensemble des oiseaux dont le bec est de la forme *Aplati*, dont la taille est comprise entre 30 cm et 50 cm, dont l'envergure est indifférente et dont la couleur du cou est soit *Roux*, soit *Orange*. Ce qui peut s'écrire, dans une syntaxe plus légère, comme :

$$(VRAI, [30, 50], ?, CouleurChaude)$$

Je me trouve devant mes quatre oiseaux que je vais examiner les uns après les autres. Ils sont représentés par le tableau suivant :

	<i>Aplati</i>	<i>Taille</i>	<i>Envergure</i>	<i>Couleur</i>	<i>Classe</i>
$e_1 =$	<i>VRAI</i>	30	49	Roux	+ (canard)
$e_2 =$	<i>FAUX</i>	70	32	Gris	- (manchot)
$e_3 =$	<i>VRAI</i>	40	46	Orange	+ (canard)
$e_4 =$	<i>FAUX</i>	60	33	Orange	- (manchot)

Je commence par le premier exemple, un canard. Comment écrire, dans le langage des concepts, une hypothèse de généralisation compatible pour le moment avec lui seul ? Pourquoi pas justement :

$$(VRAI, [30, 50], ?, CouleurChaude)$$

Mais si je veux être prudent, je peux aussi produire le concept qui se contente de mémoriser ce premier exemple :

$$(VRAI, [30, 30], [49, 49], Roux)$$

J'ai également une solution radicalement inverse, induire le concept universel :

$$(VRAI \vee FAUX) \wedge [-\infty, +\infty] \wedge [-\infty, +\infty] \wedge Couleur = (?, ?, ?, ?, ?)$$

qui signifie que j'accepte à partir de cet exemple unique de considérer comme cohérents tous les oiseaux que peut décrire mon langage des hypothèses.

Si je considère maintenant le second exemple, il est clair que cette dernière possibilité doit être éliminée puisqu'elle couvre désormais un contre-exemple. Je peux la « couper au ras » de ce contre-exemple¹ selon l'un des attributs. Il y a six solutions, parmi lesquelles :

$$v_1 = (?, [-\infty, 69], ?, ?)$$

$$v'_1 = (?, ?, ?, CouleurChaude)$$

1. C'est-à-dire en excluant avec ce contre-exemple le minimum de ce que le langage des hypothèses impose d'exclure avec lui.

Je peux aussi conserver ma politique prudente et produire :

$$v_2 = (VRAI, [30, 30], [49, 49], Roux)$$

qui est toujours cohérent vis-à-vis de mes deux exemples. J'aurais aussi pu générer, entre autres :

$$\begin{aligned} v_3 &= (?, [-\infty, 31], ?, ?) \\ v_4 &= (VRAI, [0, 35], [46, +\infty], Roux) \end{aligned}$$

Il est intuitif de vérifier que le concept v_1 est plus général que v_4 et v_2 , et que de même v_4 est plus général que v_2 . En revanche, bien que v_3 soit apparemment un concept très vaste, on ne peut pas dire qu'il soit plus général que v_4 : il contient par exemple l'objet $[VRAI, 33, 50, Roux]$ que v_4 ne contient pas. Cet exercice sera continué plus loin et la notion intuitive de concept « plus général » qu'un autre sera formalisée. Pour le moment, retenons quelques principes qui gouvernent cette façon de procéder :

- Une formule du langage des concepts décrit un ensemble d'objets (ici, d'oiseaux).
- Les concepts sont reliés entre eux par une relation de généralité, qui reflète l'inclusion des ensembles d'objets qu'ils représentent.
- Le nombre de concepts cohérents avec les exemples est très grand (ou infini, si la taille des objets n'est pas mesurée en nombres entiers).
- Les exemples sont introduits les uns après les autres et à chaque fois l'ensemble courant d'hypothèses se modifie pour qu'elles restent cohérentes avec les exemples déjà vus.

Ce chapitre a pour objet de formaliser ces notions, en particulier grâce à l'introduction d'une relation d'ordre dans l'espace des concepts. Il va aussi montrer qu'il est possible de calculer, en traitant séquentiellement les exemples, deux ensembles finis S et G de concepts à partir desquels on peut déduire tous ceux qui acceptent les exemples et refusent les contre-exemples.

Notations utiles pour le chapitre

\mathcal{H}	L'ensemble de toutes les hypothèses, choisi par l'apprenant
\mathcal{L}_X	Le langage de description des exemples
$\mathcal{L}_{\mathcal{H}}$	Le langage de description des hypothèses
$\text{couverture}(h)$	L'ensemble des exemples couverts par l'hypothèse h
\preceq	... est plus spécifique que ...
\succeq	... est plus général que ...
\top	L'élément maximal de \mathcal{H}
\perp	L'élément minimal de \mathcal{H}
hps	L'hypothèse la plus spécifique cohérente avec un ensemble d'exemples
hpg	L'hypothèse la plus générale cohérente avec un ensemble d'exemples
\vee	La disjonction logique
\wedge	La conjonction logique
$\stackrel{\text{gen}}{\Rightarrow}$	Un opérateur de généralisation
$\stackrel{\text{spe}}{\Rightarrow}$	Un opérateur de spécialisation

4.1 Les concepts de base

4.1.1 La description des attributs, la description des concepts

Nous disposons d'un ensemble d'apprentissage de m exemples $\mathcal{S} = \{(\mathbf{x}_i, \mathbf{u}_i), i = 1, m\}$, avec \mathbf{u}_i valant soit *VRAI* ou + (exemple du concept à apprendre), soit *FAUX* ou - (contre-exemple). L'ensemble des exemples positifs est noté \mathcal{S}_+ , celui des exemples négatifs \mathcal{S}_- . Les valeurs \mathbf{x} sont prises dans un espace de représentation² \mathcal{X} , qui est une combinaison d'attributs de nature diverse. Nous considérons ici que les attributs peuvent être :

- *Binaires* : le bec d'un oiseau est *Aplat* ou non.
- *Numériques* comme l'*Envergure*.
- *Nominaux* comme la *Couleur*. En général, un attribut nominal est simplement une énumération de valeurs possibles comme *Couleur* = {Rouge, Orange, Gris, Noir}. Cet ensemble de valeurs possibles s'appelle un domaine. Mais une hiérarchie peut exister sur ces valeurs, comme dans notre exemple d'introduction :

CouleurChaud = {Rouge, Orange},

CouleurFroide = {Gris, Noir} et

Couleur = {*CouleurChaud*, *CouleurFroide*}.

On sait dans ce cas que l'attribut est appelé *arborescent* (voir le chapitre 3 et la figure 4.1).

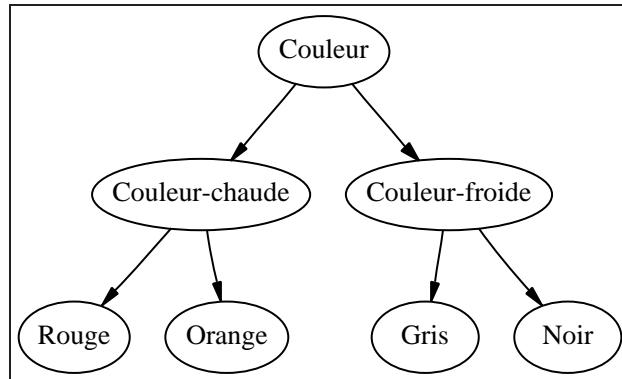


FIG. 4.1 – L'attribut *Couleur* est arborescent.

On cherche à apprendre des hypothèses, ou concepts, qui sont des éléments écrits dans $\mathcal{L}_{\mathcal{H}}$, le langage des hypothèses³ (voir le chapitre 1). $\mathcal{L}_{\mathcal{H}}$ peut être défini de façons variées, mais nous pouvons déjà donner deux exemples : le premier langage est la logique des propositions sur des sélecteurs, qui est apparu informellement dans l'exemple d'introduction ; le second langage permet de décrire des unions de deux rectangles dans le plan discret. Il sera explicité un peu plus loin.

4.1.2 Les sélecteurs

Il est fréquent dans l'apprentissage par l'espace de versions que le langage $\mathcal{L}_{\mathcal{H}}$ choisi pour représenter les concepts soit, comme dans l'exemple liminaire, une partie de la logique des

2. L'espace de représentation est noté *LI* (pour *Language of Instances*) dans la terminologie originale de l'espace des versions.

3. Dans le vocabulaire classique de l'espace des versions, le langage des hypothèses est souvent noté *LG* (pour *Language of Generalizations*).

propositions. Plus précisément, il s'agit souvent d'une conjonction de propriétés binaires sur les attributs des exemples. On appelle *sélecteurs* ces propriétés, que l'on peut définir ainsi :

Définition 4.1

Un sélecteur est une application agissant sur un seul attribut de l'espace de représentation des données, à valeurs dans {VRAI, FAUX}.

Selon la nature des attributs, un sélecteur prend des formes différentes :

- Si l'attribut est binaire, le sélecteur s'écrit (*attribut = VRAI*) ou (*attribut = FAUX*) ; sa valeur *VRAI* ou *FAUX* se déduit directement de celle de l'attribut.
- Si l'attribut est nominal de domaine D , le sélecteur s'écrit (*attribut ∈ D'*), avec $D' ⊂ D$. Il est *VRAI* si l'attribut prend une valeur de D' , *FAUX* sinon.
- Si l'attribut est arborescent, le sélecteur s'écrit (*attribut = V*), où V est une valeur attachée à un noeud de l'arbre. Il est *VRAI* si la valeur de l'attribut est un noeud compris au sens large entre V et une feuille de l'arbre.
- Si l'attribut est numérique, le sélecteur est défini par un intervalle de \mathbb{R} . Il est *VRAI* si la valeur est incluse dans cet intervalle, bornes comprises.

Pour revenir à notre exemple de départ, le concept :

$$[Aplati = VRAI] \wedge Hauteur \in [30, 50] \wedge Largeur \in [-\infty, +\infty] \wedge [Couleur = CouleurChaud]$$

qui s'écrit de manière simplifiée :

$$(VRAI, [30, 50], ?, CouleurChaud)$$

est composé d'une conjonction de quatre sélecteurs, un pour chaque attribut. L'attribut *Aplati* est binaire, *Taille* et *Envergure* sont numériques et *Couleur* est arborescent, comme le montre la figure 4.1. Sur l'objet :

Bec Aplati	Taille	Envergure	Couleur du cou
VRAI	60	46	Noir

le premier sélecteur est *VRAI*, le second est *FAUX*, le troisième est *VRAI* et le quatrième est *FAUX*.

4.1.3 La relation de généralité entre les hypothèses

Nous avons vu dans la section 1.5, que l'induction supervisée pouvait être considérée comme un jeu entre l'espace des exemples et l'espace des hypothèses. Le processus d'apprentissage teste les hypothèses candidates de \mathcal{H} sur les exemples d'apprentissage dans \mathcal{X} . Les informations ainsi glanées servent à déterminer d'autres hypothèses candidates, et ainsi de suite jusqu'au critère d'arrêt.

Nous supposons ici que les exemples d'apprentissage sont considérés séquentiellement, donc qu'à l'étape t l'apprenant a fabriqué une hypothèse candidate h_t cohérente avec l'échantillon d'apprentissage partiel \mathcal{S}_t . Un nouvel exemple d'apprentissage $\mathbf{z}_{t+1} = (\mathbf{x}_{t+1}, \mathbf{u}_{t+1})$, avec $\mathbf{u}_{t+1} \in \{VRAI, FAUX\}$ devient alors disponible. Il y a deux possibilités: soit il est correctement classé par l'hypothèse courante h_t , donc $h_t(\mathbf{x}_{t+1}) = \mathbf{u}_{t+1}$, auquel cas il n'y a pas de raison de modifier h_t et l'on a simplement $h_{t+1} = h_t$, soit \mathbf{x}_{t+1} n'est pas correctement classé par h_t . Deux cas sont alors possibles.

- L'exemple est de classe négative, c'est un contre-exemple du concept cible, et il est incorrectement classé comme positif par h_{t+1} . Cela signifie que la partie de \mathcal{X} « couverte »

par h_{t+1} est trop grande, au moins en ce qui concerne le point \mathbf{x}_{t+1} , (donc que le concept courant n'est plus correct). Il faut donc la réduire, c'est-à-dire chercher une sous-partie excluant \mathbf{x}_{t+1} mais couvrant tous les exemples positifs de \mathcal{S}_t .

- Au contraire, \mathbf{x}_{t+1} est de classe positive, et il est incorrectement classé comme négatif par h_{t+1} . Dans ce cas, cela signifie que la partie de \mathcal{X} « couverte » par h_{t+1} est trop petite, au moins en ce qui concerne le point \mathbf{x}_{t+1} (donc que le concept courant n'est plus complet). Il faut donc l'augmenter, c'est-à-dire chercher une sur-partie incluant \mathbf{x}_{t+1} mais ne couvrant aucun des exemples négatifs de \mathcal{S}_t (voir la figure 4.2).

Dans les deux cas, il est patent que l'hypothèse courante doit être modifiée en fonction des relations d'inclusion dans \mathcal{X} . Il faut donc trouver une relation entre les hypothèses dans \mathcal{H} qui respecte la relation d'inclusion dans \mathcal{X} . On parle de *relation de généralité* entre les hypothèses.

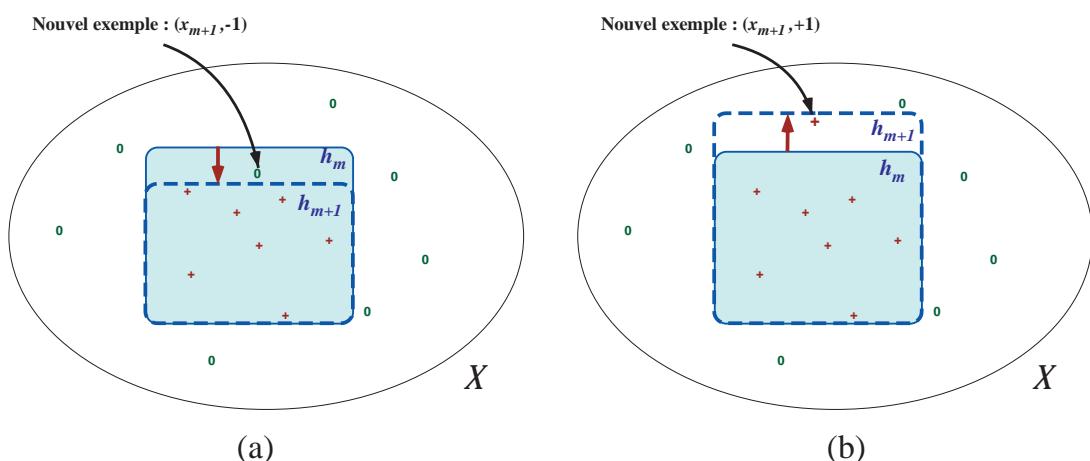


FIG. 4.2 – Lorsqu'un nouvel exemple est mal classé par l'hypothèse courante, il faut modifier celle-ci soit en la réduisant au sens de l'inclusion (a) afin d'exclure le nouvel exemple s'il est négatif, soit en l'augmentant (b) s'il est positif.

Plus formellement, nous dirons qu'une hypothèse h_1 est plus spécifique ou encore moins générale qu'une hypothèse h_2 si et seulement si l'ensemble des exemples couverts par h_1 est inclus dans l'ensemble des exemples couverts par h_2 .

Définition 4.2 (Couverture d'une hypothèse)

La couverture d'une hypothèse $h \in \mathcal{H}$, notée $\text{couverture}(h)$, est l'ensemble des exemples de \mathcal{X} que décrit h . On dit que h couvre les éléments de $\text{couverture}(h)$.

Définition 4.3 (Relation de généralité dans \mathcal{H})

Une hypothèse h_1 est plus spécifique⁴ (ou moins générale) qu'une hypothèse h_2 , ce qui se note $h_1 \preceq h_2$, si et seulement si $\text{couverture}(h_1) \subseteq \text{couverture}(h_2)$.

Les relations \preceq et \succeq sur \mathcal{H} sont illustrées dans les figures 4.3 et 4.4.

4. Cette relation est à prendre au sens large : toute hypothèse est plus spécifique qu'elle-même.

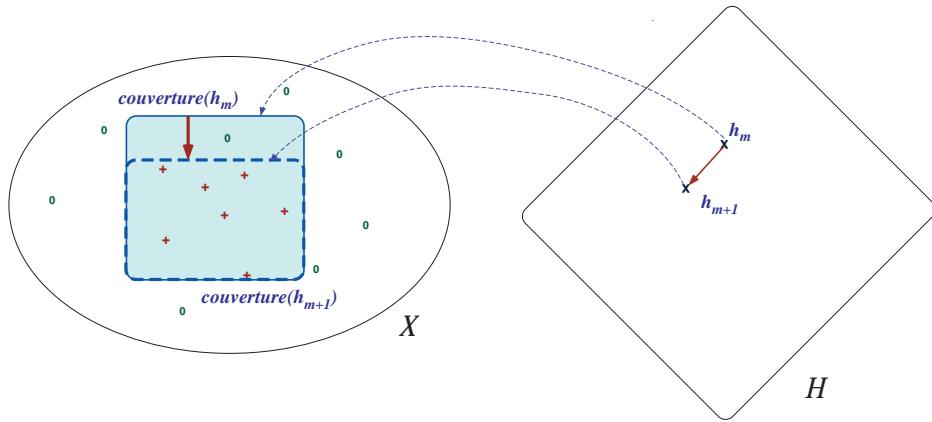


FIG. 4.3 – La relation d'inclusion dans \mathcal{X} induit la relation de généralisation dans \mathcal{H} . Ici, $h_{m+1} \preceq h_m$.

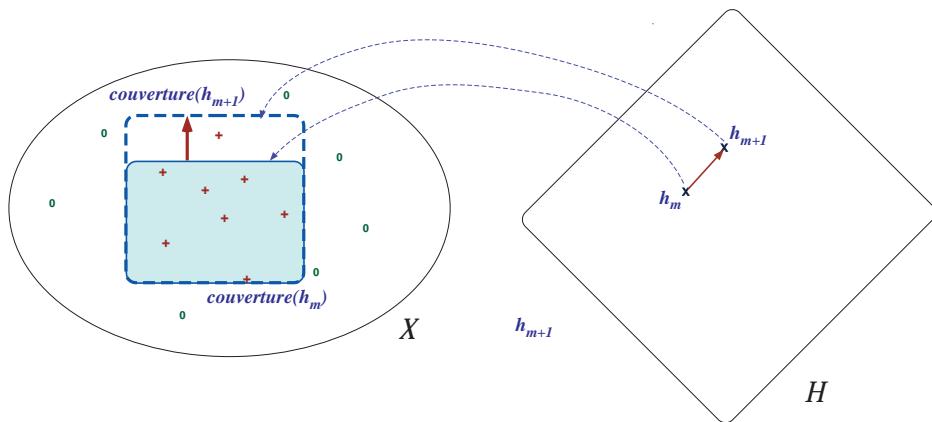


FIG. 4.4 – La relation d'inclusion dans \mathcal{X} induit la relation de généralisation dans \mathcal{H} . Ici, $h_{m+1} \succeq h_m$.

4.1.4 La relation entre un objet et un concept

Nous nous situons dans ce chapitre dans le cadre du principe inductif *ERM*, qui a été décrit au chapitre 2. En pratique, cela signifie que chaque hypothèse sera évaluée en fonction de sa performance mesurée sur l'échantillon d'apprentissage, c'est-à-dire de son *risque empirique*. Un cas particulier d'induction est celui où l'on cherche une hypothèse s'accordant parfaitement aux données, ne faisant aucune erreur sur les exemples d'apprentissage. Le risque empirique est alors mesuré sur chaque exemple par la fonction de perte binaire :

$$l(\mathbf{u}_i, h(\mathbf{x}_i)) = \begin{cases} 0 & \text{si } \mathbf{u}_i = h(\mathbf{x}_i) \\ 1 & \text{si } \mathbf{u}_i \neq h(\mathbf{x}_i) \end{cases}$$

Le risque empirique sur l'échantillon d'apprentissage \mathcal{S} s'écrit alors :

$$R_{Emp}(h) = \sum_{\mathcal{S}} l(\mathbf{u}_i, h(\mathbf{x}_i))$$

Il mesure le nombre d'erreurs commises par l'hypothèse h sur l'échantillon d'apprentissage.

Définition 4.4 (Hypothèse cohérente, correcte, complète)

Lorsque le risque empirique associé à une hypothèse est nul, on dit que l'hypothèse est cohérente⁵. Cela signifie :

1. *Que tous les exemples positifs de l'échantillon d'apprentissage sont correctement étiquetés par l'hypothèse : on dit aussi que l'hypothèse « couvre » tous les exemples positifs. L'hypothèse est alors dite complète.*
2. *Que tous les exemples négatifs sont correctement classés par l'hypothèse, c'est-à-dire rejetés comme ne faisant pas partie du concept. On dit alors que l'hypothèse ne couvre pas les exemples négatifs : elle est dite correcte.*

Si l'on suppose que les exemples étiquetés sont issus d'un concept cible, un risque empirique nul signifie que, sur l'échantillon d'apprentissage au moins, le concept cible et l'hypothèse considérée coïncident.

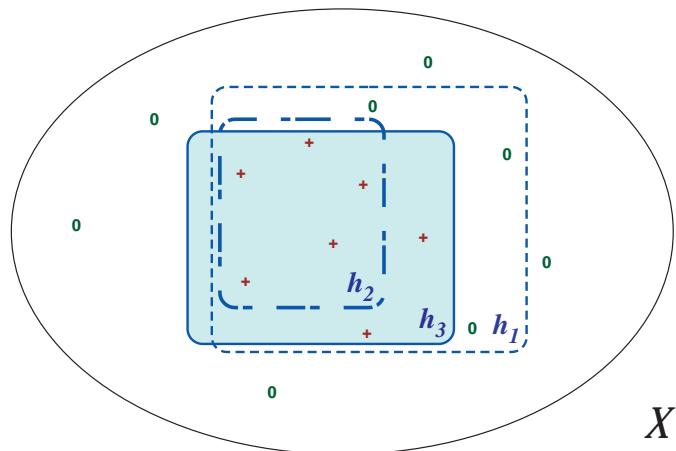


FIG. 4.5 – *La notion de couverture des exemples (figurés par des « + ») et des contre-exemples (figurés par des « 0 ») par les hypothèses. Les hypothèses sont ici figurées par le sous-espace des exemples qu'elles couvrent dans l'espace \mathcal{X} . Les hypothèses h_1 , h_2 et h_3 sont respectivement complète mais incorrecte, correcte mais incomplète, et complète et correcte, c'est-à-dire cohérente.*

4.2 La structuration de l'espace des hypothèses

4.2.1 Préliminaires

Dans sa version de base, la méthode de l'espace des versions a pour but de résoudre le problème suivant :

trouver tous les éléments de \mathcal{H} cohérents avec les exemples.

Comme on le verra, la relation de généralité sur \mathcal{H} permet d'éviter une énumération complète de ses individus, tout en gardant la possibilité de vérifier pour tout concept s'il est cohérent ou non avec les données d'apprentissage.

5. Nous évitons ici l'anglicisme « consistant » qui est souvent employé. Il se confond souvent avec la « consistance » de la méthode *ERM*, pour laquelle nous employons le mot « pertinence » (voir le chapitre 2).

Il est à noter que cette méthode de base exclut le bruit dans les données d'apprentissage, c'est-à-dire qu'elle ne sait pas traiter le cas où deux objets identiques sont l'un étiqueté positif, l'autre étiqueté négatif.

Il est commode de supposer que le langage de généralisation inclut le langage de représentation des exemples : ainsi, chaque exemple peut être considéré comme une hypothèse ne couvrant que cet exemple, qui s'écrit comme l'exemple lui-même. C'est ce que l'on appelle *l'astuce de la représentation unique (single representation trick)*. Ce n'est pas tout à fait le cas du problème d'apprentissage donné en introduction, puisque l'exemple :

	<i>Aplat</i>	<i>Taille</i>	<i>Envergure</i>	<i>Couleur</i>	<i>Classe</i>
$e_1 =$	<i>VRAI</i>	30	49	Roux	+

s'écrit dans le langage des concepts de manière un peu différente :

$$(VRAI, [30, 30], [49, 49], Roux)$$

Mais la correspondance est assez directe pour que l'on puisse ignorer la fonction de transformation. Il existe des cas où cette transformation est plus complexe⁶.

Pour simplifier, nous supposerons dans la suite sans réelle perte de généralité que

$$\mathcal{L}_X \subset \mathcal{L}_H \quad (4.1)$$

\mathcal{L}_X est donc identifié à un sous-ensemble de \mathcal{L}_H , ce que l'on peut exprimer d'une autre façon : un exemple est le concept qui généralise le moins l'exemple en question.

4.2.2 Un exemple : les paires de rectangles

Supposons que l'on cherche à apprendre un concept défini comme : « soit un rectangle, soit une paire de rectangles, à coordonnées entières et aux côtés parallèles aux axes ». Les objets de l'espace de représentation sont des points du plan \mathbb{N}^2 , définis par leurs deux coordonnées entières. Un objet est couvert par un concept quand le point correspondant est à l'intérieur de la surface (frontière comprise) définie par l'union des deux rectangles. La relation de généralité entre les concepts (les paires de rectangles) correspond naturellement à l'inclusion géométrique dans l'espace des exemples.

L'ensemble d'apprentissage est un ensemble de points du plan, certains marqués +, les autres -. On cherche à construire toutes les surfaces définies par l'intérieur d'un rectangle ou de l'union de deux rectangles qui incluent les points exemples positifs et excluent les points négatifs.

Un concept s se représente donc en général par une formule du type :

$$s = [3, 5] \times [1, +\infty] \cup [2, 7] \times [2, 3]$$

autrement dit une phrase dans un langage des concepts \mathcal{L}_H . La signification de s est la suivante : il est l'union de deux rectangles du plan (x_1, x_2) . Le premier est compris entre les valeurs 3 et 5 sur l'axe x_1 et les valeurs 1 et $+\infty$ sur l'axe x_2 , le second est compris entre les valeurs 2 et 7 sur l'axe x_1 et les valeurs 2 et 3 sur l'axe x_2 (voir la figure 4.6).

La description d'un exemple dans ce langage, autrement dit la définition de l'inclusion $\mathcal{L}_X \subset \mathcal{L}_H$, se fait simplement en considérant qu'un point est un rectangle de côtés de longueur nulle. C'est ainsi que les points

$$\mathbf{a} = (4, 8) \text{ et } \mathbf{b} = (6, 2)$$

6. C'est le cas de l'inférence de grammaires régulières que nous verrons au chapitre 7. Dans ce problème, un exemple est représenté par une séquence de lettres et son apprentissage par cœur par un automate qui ne reconnaît que cette séquence. C'est encore plus net en programmation logique inductive (chapitre 5), où la vérification qu'un concept couvre un exemple se fait par l'exécution d'un programme Prolog, ce qui est loin d'être trivial.

qui peuvent s'écrire dans le langage \mathcal{L}_H comme les phrases :

$$\mathbf{a} = [4, 4] \times [8, 8] \text{ et } \mathbf{b} = [6, 6] \times [2, 2]$$

appartiennent au concept s .

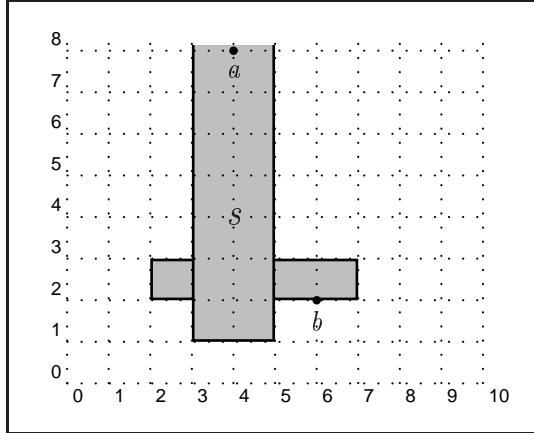


FIG. 4.6 – Le concept $s = [3, 5] \times [1, +\infty] \cup [2, 7] \times [2, 3]$ couvre les deux points $\mathbf{a} = (4, 8)$ et $\mathbf{b} = (6, 2)$. Les points sur la frontière de l'un des deux rectangles sont considérés comme appartenant au concept.

4.2.3 Un ordre partiel sur l'espace des hypothèses

La relation d'inclusion définie sur \mathcal{X} induit une relation de généralité sur \mathcal{H} qui est une relation d'ordre partiel. La figure 4.7 illustre cette notion. Cette relation est partielle et non pas totale, ce qui signifie que deux éléments quelconques dans l'espace considéré peuvent ne pas être liés par cette relation.

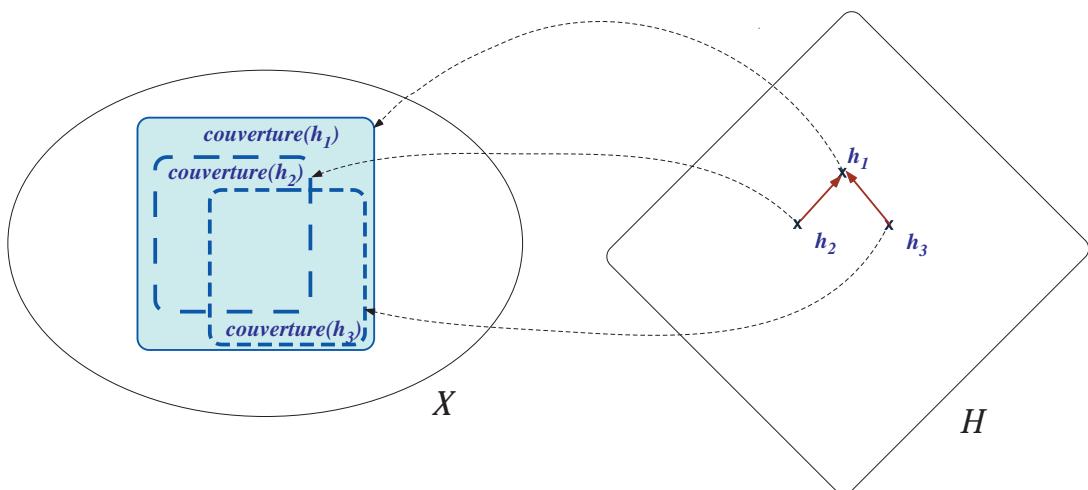


FIG. 4.7 – La relation d'inclusion dans \mathcal{X} induit la relation de généralisation dans \mathcal{H} . Il s'agit d'une relation d'ordre partiel : ici, les hypothèses h_2 et h_3 sont incomparables entre elles, mais elles sont toutes les deux plus spécifiques que h_1 .

Une relation d'ordre partiel induit une structure de *treillis* sur \mathcal{H} . Cela signifie que pour tout couple d'hypothèses h_i et h_j , il existe au moins une hypothèse qui soit plus générale que chacune d'entre elles et qu'il n'est pas possible de la spécifier sans perdre cette propriété. L'ensemble de ces hypothèses est appelé le *généralisé maximalement spécifique* de h_i et h_j et noté $gms(h_i, h_j)$. De même, il existe un ensemble d'hypothèses plus spécifiques que h_i et h_j qu'il n'est pas possible de généraliser sans perdre cette propriété. On appelle cet ensemble le *spécialisé maximalement général* et on le note $smg(h_i, h_j)$.

Par une extension facile au cas de plus de deux hypothèses, on peut définir de même un ensemble $gms(h_i, h_j, h_k, \dots)$ et un ensemble $smg(h_i, h_j, h_k, \dots)$.

Finalement, nous supposons⁷ qu'il existe dans \mathcal{H} une hypothèse plus générale que toutes les autres (ou élément maximal) notée \top et une hypothèse plus spécifique que toutes les autres (ou élément minimal) notée \perp (voir la figure 4.8).

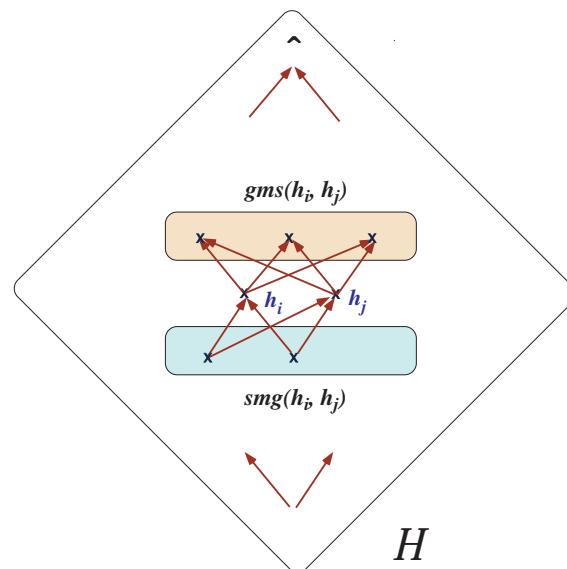


FIG. 4.8 – Une vision schématique et partielle du treillis de généralisation sur \mathcal{H} induit par la relation d'inclusion dans \mathcal{X} . Chaque flèche indique la relation de généralité (notée \preceq dans le texte).

Les exemples ainsi que plusieurs figures du chapitre 1 et de ce chapitre montrent clairement que la relation d'inclusion est fondamentale pour le problème de l'induction. En effet, une hypothèse incorrecte (donc couvrant indûment des exemples négatifs) devra être spécialisée pour que sa couverture exclut ces exemples, alors qu'une hypothèse incomplète (ne couvrant pas tous les exemples positifs connus) devra être généralisée pour que ces exemples deviennent éléments de sa couverture. Il est donc naturel que le processus d'induction soit guidé par ces relations d'inclusion.

Nous avons également déjà souligné dans le chapitre 1 que l'induction nécessite une mise à jour des hypothèses directement dans \mathcal{H} . Comme l'espace \mathcal{H} des hypothèses est défini par son langage de description $\mathcal{L}_{\mathcal{H}}$, cela signifie qu'il faut trouver comment associer à la relation d'inclusion dans \mathcal{X} des opérations syntaxiques sur $\mathcal{L}_{\mathcal{H}}$ correspondant à la relation de généralité. Trouver des équivalences aux relations d'inclusion dans \mathcal{X} revient donc à trouver des opérateurs

7. C'est en général valide.

dans le langage \mathcal{L}_H qui permettent de modifier une hypothèse h_m en une nouvelle hypothèse h_{m+1} inclue dans la première ou l'incluant⁸, c'est-à-dire plus spécifique ou plus générale.

4.2.4 Quelques opérateurs de spécialisation et de généralisation

Le problème de la recherche d'opérateurs syntaxiques de généralisation ou de spécialisation sera également débattu dans le chapitre 5 portant sur la programmation logique inductive, en particulier parce que la solution n'est pas évidente lorsque l'on utilise des représentations en logique des prédictats dite aussi logique d'ordre 1. Il est en revanche facile d'exhiber des exemples d'opérateurs satisfaisants dans le cas de représentations fondées sur la logique des propositions et la représentation attribut-valeur (voir le chapitre 11). À titre d'illustration, nous présentons quelques-uns de ces opérateurs de généralisation.

L'opération de généralisation est notée $\xrightarrow{\text{gen}}$. Une formule du type $A \wedge (B = v_1) \in \mathcal{C}$ signifie qu'un objet couvert par le concept \mathcal{C} est décrit par la conjonction d'un sélecteur sur les attributs A (avec A binaire) et B (avec B nominal ordonné), ce dernier valant v_1 pour B . On constatera que les généralisations proposées ne sont évidemment pas des opérations logiquement valides.

Opérateur de clôture d'intervalle

$$\left. \begin{array}{l} A \wedge (B = v_1) \in \mathcal{C} \\ A \wedge (B = v_2) \in \mathcal{C} \end{array} \right\} \xrightarrow{\text{gen}} A \wedge (B \in [v_1, v_2]) \in \mathcal{C}$$

Par exemple:

$$\left. \begin{array}{l} \text{Bec aplati} \wedge (\text{envergure} = 50) \in \text{canard} \\ \text{Bec aplati} \wedge (\text{envergure} = 55) \in \text{canard} \end{array} \right\} \xrightarrow{\text{gen}} \text{Bec aplati} \wedge (\text{envergure} \in [50, 55]) \in \text{canard}$$

Opérateur de l'ascension dans l'arbre de hiérarchie

Pour généraliser une description incluant un attribut arborescent, il suffit de le remplacer par l'un de ses descendants dans l'arbre :

$$\left. \begin{array}{l} A \wedge (B = n_1) \in \mathcal{C} \\ A \wedge (B = n_2) \in \mathcal{C} \end{array} \right\} \xrightarrow{\text{gen}} A \wedge (B = N) \in \mathcal{C}$$

où N est le plus petit noeud ascendant commun aux noeuds n_1 et n_2 . Par exemple :

$$\left. \begin{array}{l} \text{Bec aplati} \wedge (\text{couleur} = \text{Roux}) \in \text{canard} \\ \text{Bec aplati} \wedge (\text{couleur} = \text{Orange}) \in \text{canard} \end{array} \right\} \xrightarrow{\text{gen}} \text{Aplati} \wedge (\text{couleur} = \text{couleur-chaude}) \in \text{canard}$$

Opérateur d'abandon de conjonction

$$A \wedge B \in \mathcal{C} \xrightarrow{\text{gen}} A \in \mathcal{C}$$

Par exemple :

$$\text{Bec Aplati} \wedge (\text{couleur} = \text{Roux}) \in \text{canard} \xrightarrow{\text{gen}} \text{Bec Aplati} \in \text{canard}$$

8. On tire profit ici de la confusion assumée entre la notion de concept et celle de partie de \mathcal{X} (voir l'équation 4.1) pour parler de concept inclus dans un autre, alors que la relation d'inclusion n'a, à proprement parler, de sens que pour les catégories qui sont définies sur \mathcal{X} .

Opérateur d'ajout d'alternative

$$A \in \mathcal{C} \xrightarrow{\text{gen}} A \vee B \in \mathcal{C}$$

Par exemple:

$$\text{Bec Aplat} \in \text{canard} \xrightarrow{\text{gen}} \text{Bec Aplat} \vee \text{Couleur} = \text{Orange} \in \text{canard}$$

Opérateur de changement de conjonction en disjonction

$$A \wedge B \in \mathcal{C} \xrightarrow{\text{gen}} A \vee B \in \mathcal{C}$$

Par exemple:

$$(\text{Bec Aplat} \wedge \text{Couleur} = \text{Orange} \in \text{canard}) \xrightarrow{\text{gen}} (\text{Bec Aplat} \vee \text{Couleur} = \text{Orange} \in \text{canard})$$

Chaque opérateur de généralisation permet de transformer l'expression d'une hypothèse en l'expression d'une hypothèse plus générale, couvrant davantage d'éléments de l'espace des exemples \mathcal{X} . Il est possible de renverser chaque opérateur de généralisation pour obtenir des opérateurs de spécialisation qui transforment une hypothèse en une hypothèse moins générale, ou plus spécifique, couvrant moins d'éléments de \mathcal{X} .

Ainsi l'opérateur de généralisation par montée dans la hiérarchie des descripteurs peut fournir un opérateur de spécialisation par descente dans la hiérarchie :

$$A \wedge (B = N) \in \mathcal{C} \xrightarrow{\text{spe}} A \wedge (B = n_1) \in \mathcal{C}$$

où N est un nœud ascendant du descripteur n_1 .

$$\text{Bec Aplat} \wedge (\text{Couleur} = \text{couleur-chaude}) \in \text{canard}$$

$$\xrightarrow{\text{spe}} \text{Bec Aplat} \wedge (\text{Couleur} = \text{Roux}) \in \text{canard}$$

Pour résumer, si un espace d'hypothèses \mathcal{H} est défini par un langage $\mathcal{L}_{\mathcal{H}}$ qui admet des opérateurs de spécialisation et de généralisation, alors il est muni d'une structure d'ordre partiel associée à la relation d'inclusion sur l'espace des exemples \mathcal{X} . De ce fait, elle est particulièrement pertinente pour la tâche d'induction. Ainsi, la recherche d'une hypothèse cohérente avec les exemples d'apprentissage peut être guidée par ces opérateurs et être par conséquent beaucoup plus efficace qu'une recherche par gradient (chapitre 3). En examinant quelques propriétés de l'espace \mathcal{H} ainsi muni d'une relation d'ordre, nous allons voir que l'avantage peut être plus important encore.

4.2.5 Quelques propriétés utiles d'un espace structuré par une relation d'ordre partiel

Rappelons d'abord qu'il n'y a généralement pas bijection entre l'espace des hypothèses \mathcal{H} et l'espace des exemples \mathcal{X} : c'est ce qu'implique l'existence d'un biais de représentation, c'est-à-dire les limites de l'expressivité du langage $\mathcal{L}_{\mathcal{H}}$. De ce fait, il est important de s'assurer que certaines propriétés sont vérifiées. Il en est deux qui nous concernent spécialement. La première a trait à la convexité de \mathcal{H} : est-on certain que l'application des opérateurs de spécialisation/généralisation sur des expressions de $\mathcal{L}_{\mathcal{H}}$ produit toujours des expressions valides qui ont une contrepartie dans \mathcal{X} ? En d'autres termes, ne risque-t-on pas, en jouant avec ces opérateurs, de produire des

« hypothèses » qui n'auraient pas de sens ? La deuxième propriété est duale de la précédente : peut-il y avoir des hypothèses dans \mathcal{H} qui sont de fait plus générales ou plus spécifiques qu'une autre hypothèse de \mathcal{H} , mais qu'on ne puisse pas obtenir à partir de celle-ci par une séquence d'opérateurs de spécialisation/généralisation ? Si l'une ou l'autre de ces deux propriétés s'avérait non vérifiée, alors l'exploration de \mathcal{H} par l'application des opérateurs pourrait conduire à des résultats aberrants : soit des hypothèses sans signification dans \mathcal{X} , soit au contraire la non production d'hypothèses pertinentes de \mathcal{H} . Heureusement, ces deux propriétés peuvent être obtenues. Plus formellement :

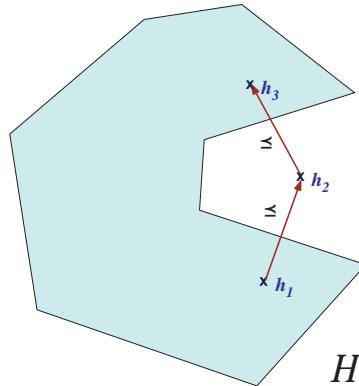


FIG. 4.9 – Un exemple d'ensemble non convexe pour la relation de généralité.

Définition 4.5 (Ensemble convexe pour la généralisation)

Un ensemble E dont les éléments sont représentables dans un langage $\mathcal{L}_\mathcal{H}$ est convexe si et seulement si :

Pour tous h_1, h_2, h_3 tels que $h_1, h_3 \in E$, et $h_1 \preceq h_2 \preceq h_3$ alors $h_2 \in E$

Propriété 4.1 (Théorème de convexité ([Hir90]))

L'ensemble des hypothèses \mathcal{H} défini par un langage $\mathcal{L}_\mathcal{H}$ sur lequel sont définis des opérateurs de spécialisation/généralisation est convexe.

Définition 4.6 (Ensemble borné pour la généralisation)

Un ensemble C d'hypothèses décrits par un langage $\mathcal{L}_\mathcal{H}$ est borné si et seulement si pour tout h dans C il existe une hypothèse g maximalement générale dans C et une hypothèse s maximalement spécifique dans C telles que $s \preceq h \preceq g$.

Il n'est pas possible de garantir la propriété d'être borné pour tout ensemble décrit sur un langage $\mathcal{L}_\mathcal{H}$ muni d'opérateurs de spécialisation/généralisation. C'est donc une contrainte qu'il faut à chaque fois vérifier ou imposer si besoin est. Cette contrainte est en général vérifiée sur les langages d'ordre 0 (logique des propositions), elle demande par contre des soins particuliers en logique d'ordre 1 (logique des prédicats).

Si les deux propriétés précédentes sont vraies, alors une troisième propriété en découle, fondamentale pour la suite.

Définition 4.7 (S : Les hypothèses cohérentes maximalement spécifiques)

L'ensemble des hypothèses de \mathcal{H} couvrant les exemples positifs et excluant les exemples négatifs,

et telles qu'il en soit pas possible de les spécialiser sans perdre ces propriétés, est appelé le S-set. Nous le noterons S dans la suite.

Définition 4.8 (G : Les hypothèses cohérentes maximalement générales)

L'ensemble des hypothèses de \mathcal{H} couvrant les exemples positifs et excluant les exemples négatifs, et telles qu'il ne soit pas possible de les généraliser sans perdre ces propriétés, est appelé le G-set. Nous le noterons G dans la suite.

Théorème 4.1 (Représentation de l'espace des versions par S et G [Hir90])

Si un ensemble d'hypothèses est convexe et borné, alors il peut être représenté par sa borne inférieure S et sa borne supérieure G .

Définition 4.9 (Espace des versions)

L'ensemble de toutes les hypothèses cohérentes avec les exemples d'apprentissage est appelé l'espace des versions.

Le théorème (4.1) prouve que l'espace des versions peut être représenté de manière économique par ses bornes S et G .

Cette propriété a trois corollaires essentiels :

1. À tout instant, l'ensemble des hypothèses cohérentes avec un ensemble d'exemples d'apprentissage est représentable par une borne inférieure et une borne supérieure : toute hypothèse comprise entre ces deux bornes est cohérente.
2. Un algorithme d'apprentissage peut opérer en calculant ces deux bornes, et donc en calculant l'ensemble des hypothèses cohérentes. Il s'agit là d'une idée novatrice par rapport aux algorithmes d'apprentissage recherchant *une* hypothèse cohérente par modification incrémentale d'une hypothèse initiale.
3. Il n'existe donc pas de concept cohérent moins spécifique qu'un élément de G ou plus spécifique qu'un élément de S .

En considérant un exemple comme une hypothèse, grâce à la relation $\mathcal{L}_x \subset \mathcal{L}_H$ (équation 4.1), nous pouvons donc remarquer que S est un sous-ensemble du *gms* de l'ensemble des exemples positifs (on ne peut rien dire d'autant simple sur G). Nous allons voir l'application de cette idée dans l'algorithme d'élimination des candidats proposé par T. Mitchell [Mit82].

4.3 La construction de l'espace des versions

4.3.1 Illustration : retour sur l'exemple des rectangles

Reprenons l'exemple des rectangles, avec les données d'apprentissage suivantes :

x_1	x_2	Classe
3	2	+
5	3	+
3	3	+
2	2	-
4	3	-

Le concept $z = [3, 3] \times [1, 6] \cup [5, 25] \times [1, 4]$ est cohérent avec ces données d'apprentissage (voir la figure 4.10).

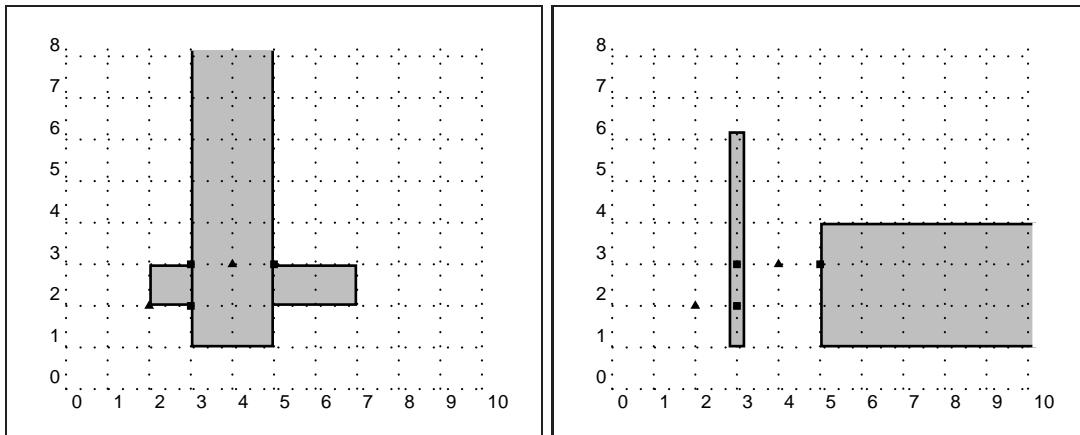


FIG. 4.10 – Le concept $s = [3, 5] \times [1, +\infty] \cup [2, 7] \times [2, 3]$ (figure de gauche) couvre tous les points d'apprentissage, positifs (carrés) comme négatifs (triangles). Il n'est donc pas cohérent avec les données d'apprentissage. En revanche le concept $z = [3, 3] \times [1, 6] \cup [5, 25] \times [1, 4]$ (figure de droite) est cohérent avec ces données : il couvre tous les exemples positifs et aucun exemple négatif.

Nous allons voir maintenant comment construire S et G à partir des exemples, et nous vérifierons que tout élément de l'espace des versions est plus spécifique qu'un certain élément de G et moins spécifique qu'un certain élément de S .

4.3.2 L'algorithme d'élimination des candidats

L'apprentissage par l'espace des versions est associé à un algorithme de construction des solutions, appelé l'*élimination des candidats* (algorithme 4.1).

Il procède de manière itérative, exemple par exemple, en mettant à jour S et G . Sa convergence est assurée par un théorème (non démontré ici) qui prouve qu'un seul examen de chaque exemple suffit et que l'ordre de présentation des exemples n'influe pas sur le résultat de l'algorithme.

Cet algorithme gère deux procédures `Généraliser(s, x, G)` et `Spécialiser(g, x, S)`, qui seront utilisées pour remplacer dans G (respectivement S) un concept devenant trop spécifique (respectivement trop général) par un ou plusieurs autres concepts permettant de respecter les contraintes de consistance. Ces procédures se définissent grâce aux notions de spécialisation minimale et de généralisation minimale. La figure 4.11 illustre les différents cas qui peuvent se rencontrer lors de la mise à jour des bornes S et G par l'algorithme d'élimination des candidats.

Il est donc démontré que cet algorithme itératif remplit le but fixé : il permet de trouver S et G . À partir de là il autorise la caractérisation de tous les concepts cohérents avec les exemples.

4.3.3 Deux exemples

4.3.3.1 Encore les rectangles

Généralisation et spécialisation minimale

Dans l'exemple des rectangles, les définitions de la spécialisation minimale et de la généralisation minimale sont faciles à comprendre intuitivement :

- Étant donnés une hypothèse et un objet, la généralisation minimale consiste à élargir l'un des deux rectangles en déplaçant le côté adéquat jusqu'à ce qu'il englobe juste l'objet. Le

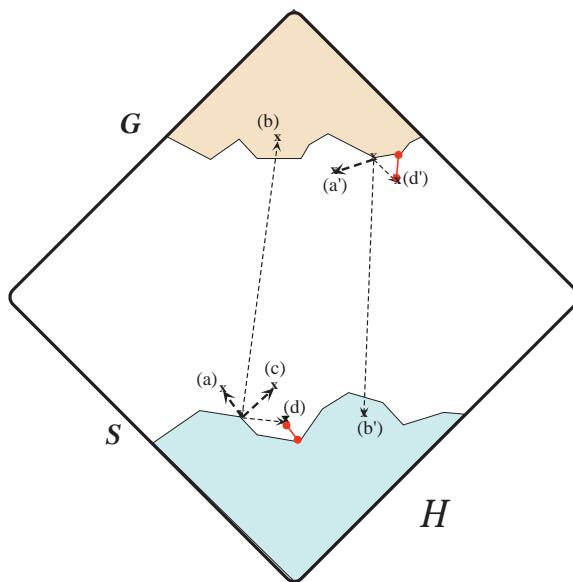


FIG. 4.11 – Cette figure schématise les différents cas possibles lors de la mise à jour des ensembles S et G par l'algorithme d'élimination des candidats. Les cas (a), (b), (c) et (d) correspondent à la mise à jour de S pour tenir compte d'un exemple positif. On suppose ici qu'un élément de S ne couvre pas ce nouvel exemple et doit être généralisé. Les flèches en pointillés illustrent le cas où quatre directions de généralisations seraient possibles. La direction (b) doit être éliminée car elle correspond à une surgénéralisation : l'hypothèse produite est en effet plus générale qu'une hypothèse de G et doit donc couvrir des exemples négatifs. L'hypothèse (d) doit également être écartée car elle est plus générale qu'une autre hypothèse de S qui est cohérente avec les exemples. Il reste donc les hypothèses (a) et (c) qui remplaceront l'ancienne hypothèse dans S . Les cas (a'), (b') et (d') illustrent des cas duals dans le cas de la mise à jour de l'ensemble G pour tenir compte d'un nouvel exemple négatif.

plus simple est de voir cette opération (qui peut se faire de deux façons) sur la figure 4.12. Il est clair que les concepts obtenus sont tous les deux plus généraux que celui dont on est parti.

- La spécialisation minimale consiste à réduire l'un des deux rectangles en déplaçant le côté adéquat jusqu'à ce l'objet se trouve juste exclu de l'hypothèse. Il y a deux ou quatre façons de s'y prendre, selon la position du point dans le concept (voir la figure 4.12).

Algorithme 4.1 Algorithme d'élimination des candidats.

Initialiser G comme l'ensemble des hypothèses les plus générales de \mathcal{H}
 Initialiser S comme l'ensemble des hypothèses les moins générales de \mathcal{H}

pour Chaque exemple x faire

si x est un exemple positif **alors**

 Enlever de G toutes les hypothèses qui ne couvrent pas x

pour Chaque hypothèse s de S qui ne couvre pas x **faire**

 Enlever s de S

Généraliser(s, x, S)

 c'est-à-dire: ajouter à S toutes les généralisations minimales h de s telles que:

- h couvre x et
- il existe dans G un élément plus général que h

 Enlever de S toute hypothèse plus générale qu'une autre hypothèse de S

fin pour

sinon

 (x est un exemple négatif)

 Enlever de S toutes les hypothèses qui couvrent x

pour Chaque hypothèse g de G qui couvre x **faire**

 Enlever g de G

Spécialiser(g, x, G)

 c'est-à-dire: ajouter à G toutes les spécialisations maximales h de g telles que:

- h ne couvre pas x et
- il existe dans S un élément plus spécifique que h

 Enlever de G toute hypothèse plus spécifique qu'une autre hypothèse de G

fin pour

fin si

fin pour

Déroulement de l'algorithme

Reprenons l'ensemble d'apprentissage:

x_1	x_2	Classe
3	2	+
5	3	+
3	3	+
2	2	-
4	3	-

Il est représenté sur la figure 4.13. Le déroulement de l'algorithme d'élimination des candidats se fait alors comme suit :

DEBUT

Initialisation

$$G_0 = [-\infty, +\infty] \times [-\infty, +\infty)$$

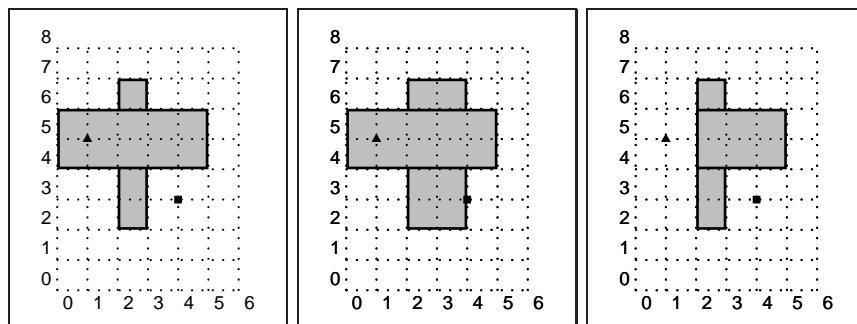


FIG. 4.12 – À gauche, un concept qui couvre un exemple négatif et ne couvre pas un exemple positif. Au centre, une généralisation minimale possible de ce concept par rapport à l'exemple positif. À droite, une spécialisation minimale possible du même concept par rapport à l'exemple négatif.

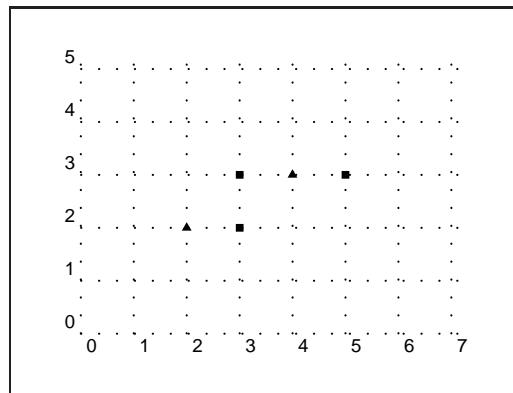


FIG. 4.13 – L'ensemble d'apprentissage. Les exemples sont représentés par des carrés et les contre-exemples par des triangles.

$$S_0 = \emptyset$$

Lecture de l'exemple positif ($x_1 = 3, x_2 = 2$)

$$G = G_0$$

$$S = \{[3, 3] \times [2, 2]\}$$

Lecture de l'exemple positif ($x_1 = 5, x_2 = 3$)

$$G = G_0$$

$$S = \{[3, 3] \times [2, 2] \cup [5, 5] \times [3, 3]\}$$

Lecture de l'exemple positif ($x_1 = 3, x_2 = 3$)

G est inchangé.

$$S = \{ [3, 3] \times [2, 2] \cup [3, 5] \times [3, 3], \\ [3, 3] \times [2, 3] \cup [5, 5] \times [3, 3] \}$$

Comme le langage des concepts est limité à l'union d'au plus deux rectangles, il y a ici une généralisation : le point ($x_1 = 4, x_2 = 3$) est admis par le premier élément de S .

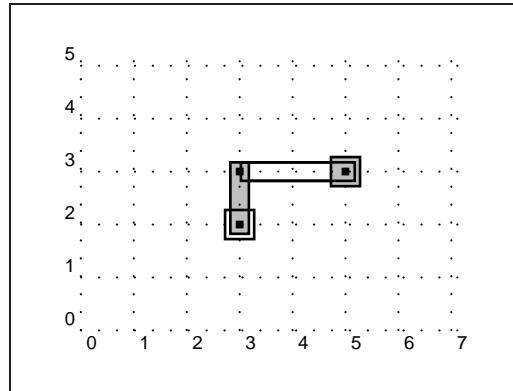


FIG. 4.14 – Après la lecture du troisième exemple positif, S possède deux éléments : la paire de rectangles grisée et celle en blanc. Tous ces rectangles ont soit une largeur nulle, soit une hauteur nulle, soit les deux. Dans ce dernier cas, ils sont réduits au point d'apprentissage.

Lecture de l'exemple négatif ($x_1 = 2, x_2 = 2$)

$$G = \{ [3, +\infty] \times [-\infty, +\infty] \cup [-\infty, +\infty] \times [3, +\infty], \\ [3, +\infty] \times [-\infty, +\infty] \cup [-\infty, +\infty] \times [-\infty, 1], \\ [-\infty, 1] \times [-\infty, +\infty] \cup [3, +\infty] \times [-\infty, +\infty] \}$$

Il y avait six façons de spécialiser G sans tenir compte de S , on en a gardé trois : les seules contenant au moins un élément de S . S est inchangé (voir les figures 4.15 et 4.16).

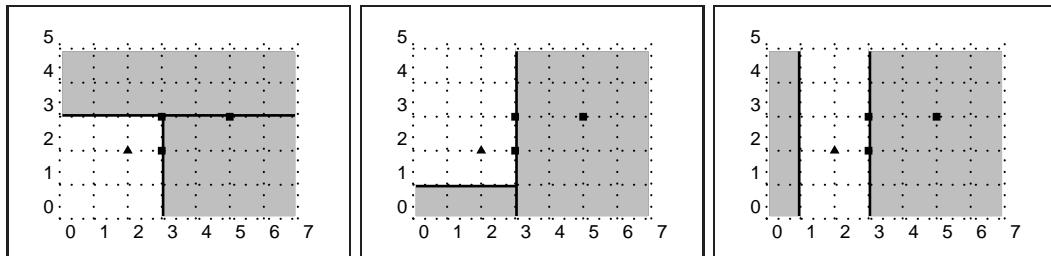


FIG. 4.15 – Après la lecture du quatrième exemple (négatif), G possède trois éléments et S est inchangé.

Lecture de l'exemple négatif ($x_1 = 4, x_2 = 3$)

$$S = \{[3, 3] \times [2, 3] \cup [5, 5] \times [3, 3]\}$$

$$\text{Notons } s \text{ l'unique élément de } S. G = \{[3, 3] \times [-\infty, +\infty] \cup [5, +\infty] \times [-\infty, +\infty]\}$$

Notons g l'unique élément de G . Le résultat est donné sur la figure 4.17.

FIN

On peut vérifier que n'importe quel concept cohérent, par exemple :

$$z = [3, 3] \times [-5, 10] \cup [5, 10] \times [1, 4]$$

ou :

$$z' = [3, 3] \times [1, 12] \cup [5, 25] \times [-1, 46]$$

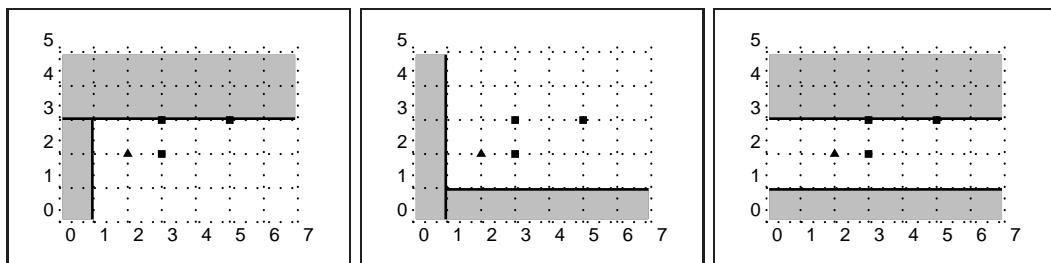


FIG. 4.16 – Trois autres possibilités d’éléments de G ont été éliminées, car elles ne couvrent aucun élément de S .

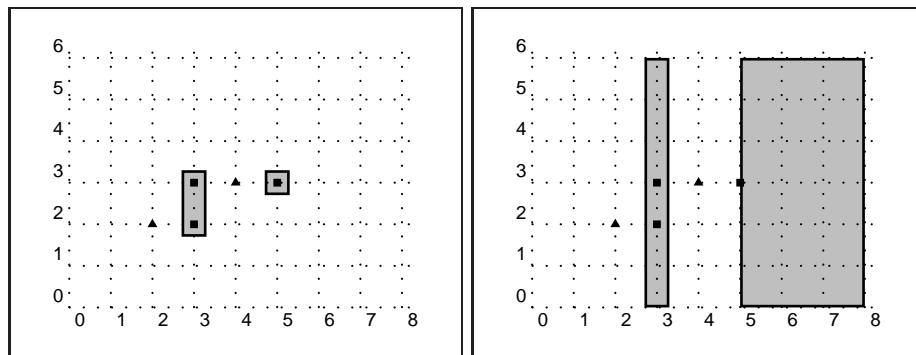


FIG. 4.17 – Après la lecture du cinquième exemple (négatif), S et G n’ont chacun qu’un seul élément : s et g .

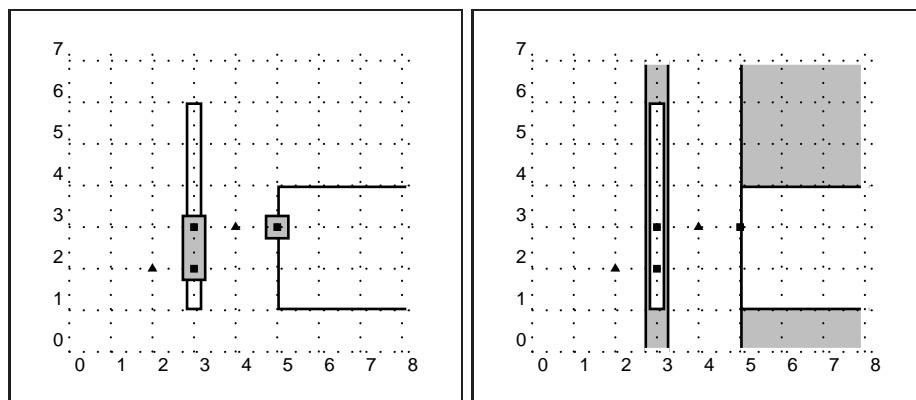


FIG. 4.18 – Deux concepts z tels que $s \succ z \succ g$

est tel que (figure 4.18) :

$$s \succ z \succ g$$

mais il n’existe pas de relation d’ordre entre z et z' .

4.3.3.2 Fonctionnement sur l’exemple des oiseaux

Généralisation et spécialisation minimale

Dans l’exemple des canards et des manchots, les opérations de spécialisation et de généralisation

sont également faciles à définir. Soit le concept :

$$v_1 = (?, [-\infty, 60], [33, +\infty], \text{CouleurChaude})$$

et le contre-exemple :

$$e_4 = (\text{FAUX}, 60, 38, \text{Orange})$$

Il y a quatre possibilités de spécialisation minimale de v_1 vis-à-vis de e_4 :

$$\begin{aligned} & (\text{VRAI} , [-\infty, 60], [33, +\infty], \text{CouleurChaude}) \\ & (?, [-\infty, 59], [33, +\infty], \text{CouleurChaude}) \\ & (?, [-\infty, 60], [39, +\infty], \text{CouleurChaude}) \\ & (?, [-\infty, 60], [33, +\infty], \text{Roux}) \end{aligned}$$

Pour le concept :

$$v_4 = (\text{VRAI}, [0, 59], [46, +\infty], \text{Roux})$$

et l'exemple :

$$(\text{FAUX}, 60, 47, \text{Orange})$$

on obtient la généralisation minimale :

$$(?, [0, 60], [47, +\infty], \text{CouleurChaude})$$

Déroulement de l'algorithme

Pour simplifier cet exemple, nous allons éliminer le troisième attribut, l'*Envergure*. Les données d'apprentissage sont donc les suivantes :

	<i>Aplat</i>	<i>Taille</i>	<i>Couleur</i>	<i>Classe</i>
$e_1 =$	<i>VRAI</i>	30	Roux	+
$e_2 =$	<i>FAUX</i>	70	Gris	-
$e_3 =$	<i>VRAI</i>	40	Orange	+
$e_4 =$	<i>FAUX</i>	60	Orange	-

DEBUT

Initialisation

$$G = (?, ?, ?)$$

$$S = \emptyset$$

Lecture de $e_1 = ((\text{VRAI}, 30, \text{Roux}), +)$

On généralise minimalement S pour couvrir e_1 :

$$S = \{ (\text{VRAI}, [30, 30], \text{Roux}) \}$$

G est inchangé: $G = (?, ?, ?)$

Lecture de $e_2 = ((\text{FAUX}, 70, \text{Gris}), -)$

S est inchangé: $S = \{ (\text{VRAI}, [30, 30], \text{Roux}) \}$

Il y a quatre spécialisations minimales de G pour rejeter e_2 :

$$\begin{aligned} & (\text{VRAI} , ?, ?) \\ & (?, [-\infty, 69] , ?) \\ & (?, [71, +\infty] , ?) \\ & (?, ?, \text{CouleurChaude}) \end{aligned}$$

On ajoute à G les trois pour lesquelles il existe dans S un élément plus spécifique.

$$G = \{ (\text{VRAI}, ?, ?), \\ (\ ?, [-\infty, 69], ?) \\ (\ ?, ?, \text{CouleurChaude}) \}$$

Lecture de $e_3 = ((\text{VRAI}, 40, \text{Orange}), +)$

On généralise S pour couvrir e_3

$$S = \{(\text{VRAI}, [30, 40], \text{CouleurChaude})\}$$

G est inchangé.

Lecture de $e_4 = ((\text{FAUX}, 60, \text{Orange}), -)$

On essaie de spécialiser chaque élément de G par rapport à e_4 .

Le premier élément ne couvre pas e_4 .

Les trois suivants donnent respectivement 3, 4 et 4 concepts possibles:

Spécialisations minimales de $(\text{VRAI}, ?, ?)$ pour rejeter $(\text{FAUX}, 60, \text{Orange})$:

$$(\text{VRAI}, [-\infty, 59], ?) \\ (\text{VRAI}, [61, +\infty], ?) \\ (\text{VRAI}, ?, \text{CouleurFroide})$$

Spécialisations minimales de $(?, [-\infty, 69], ?)$ pour rejeter $(\text{FAUX}, 60, \text{Orange})$:

$$(\text{VRAI}, [-\infty, 69], ?) \\ (\ ?, [-\infty, 59], ?) \\ (\ ? [61, +\infty], ?) \\ (\ ?, ?, \text{CouleurFroide}))$$

Spécialisations minimales de $(?, ?, \text{CouleurChaude})$ pour rejeter $(\text{FAUX}, 60, \text{Orange})$:

$$(\text{VRAI}, ?, \text{CouleurChaude}) \\ (\ ?, [-\infty, 59], \text{CouleurChaude}) \\ (\ ?, [61, +\infty], \text{CouleurChaude}) \\ (\ ?, ?, \text{Roux})$$

S est inchangé: $S = \{(\text{VRAI}, [30, 40], \text{CouleurChaude})\}$

On ajoute à G les spécialisations pour lesquelles il existe dans S un élément plus spécifique.

Finalement, en ne conservant que le concept le plus général s'il en existe deux en relation de généralité, G devient :

$$G = \{ (\text{VRAI}, ?, ?), \\ (\ ?, [-\infty, 69], ?),$$

FIN

Tout concept plus général que le seul élément de S et plus spécifique qu'un élément de G est solution. Voici trois exemples :

$$(\text{VRAI}, [9, 18], \text{Noir}) \\ (\ ?, [30, 30], \text{Orange}) \\ (\ ?, [9, 9], \text{CouleurChaude})$$

4.3.4 Un exemple d'application : le système LEX

Le système LEX a été développé par Tom Mitchell en 1983 pour apprendre automatiquement les conditions d'utilisation des opérateurs mathématiques permettant de calculer la primitive d'expressions mathématiques. C'est exactement ce à quoi les élèves des classes préparatoires aux grandes écoles passent beaucoup de temps. Au début, on leur donne un ensemble de « recettes » que l'on appelle ici des opérateurs OP . Par exemple (r et c sont ici de constantes) :

- La sortie des constantes de multiplication :

$$OP_1: \int r f(x) dx = r \int f(x) dx$$

- La règle d'intégration par parties :

$$OP_2: \int u dv = uv - \int v du$$

- La règle de l'identité :

$$OP_3: 1 \times f(x) = f(x)$$

- L'intégrale d'une somme est la somme des intégrales :

$$OP_4: \int [f_1(x) + f_2(x)] dx \rightarrow r \int f_1(x) dx + \int f_2(x) dx$$

- La primitive de la fonction $\sin(x)$:

$$OP_5: \int \sin(x) dx = -\cos(x) + c$$

- La primitive de la fonction $\cos(x)$

$$OP_6: \int \cos(x) dx = \sin(x) + c$$

- La règle d'intégration des fonctions puissance :

$$OP_7: \int x^n dx = \frac{x^{n+1}}{n+1} + c$$

Au début de leur apprentissage, les élèves connaissent ces règles mais ne savent pas exactement dans quel contexte il est judicieux d'appliquer chacune d'entre elles. De ce fait, ils mettent beaucoup de temps à résoudre les problèmes car ils se perdent dans de nombreuses impasses. Au fur et à mesure de leur entraînement, ils apprennent à appliquer ces règles juste au moment où elles font progresser vers la solution. C'est ce type d'apprentissage qu'essaie de simuler le système LEX.

Au début de son apprentissage, le système connaît un ensemble d'opérateurs d'intégration symboliques tels que ceux donnés plus haut. Il est également doté d'une taxonomie sur les concepts de fonctions mathématiques (voir figure 4.19). Le système suit alors un cycle qui est schématisé sur la figure 4.20. À chaque cycle, un module fournit un exercice à résoudre. Un système de résolution de problème tente alors de trouver une solution en enchaînant des opérateurs d'intégration. Cela fournit un arbre de résolution qui peut aboutir à une solution

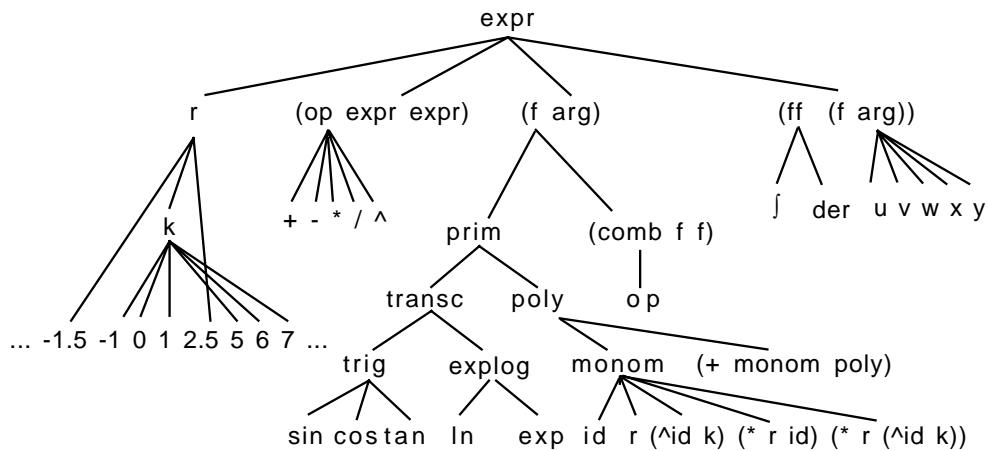


FIG. 4.19 – Une taxonomie des fonctions de base telle que celle employée par le système LEX.

ou à un échec (par exemple si le système ne trouve pas de solution avec les ressources calcul allouées). Chaque utilisation des opérateurs selon une branche ayant mené à un succès fournit un exemple positif de l'emploi de cet opérateur (en fait il faut aussi examiner si la solution est optimale). Inversement, chaque utilisation d'un opérateur le long d'une branche ayant mené à un échec correspond à une utilisation erronée de cet opérateur et fournit donc un exemple négatif de contexte d'utilisation. Ces exemples et contre-exemples sont alors fournis à un système d'apprentissage utilisant l'algorithme d'élimination des candidats qui calcule ainsi les bornes S et G définissant les contextes dans lesquels il est approprié d'utiliser l'opérateur considéré (voir la figure 4.21 pour un exemple de cycle).

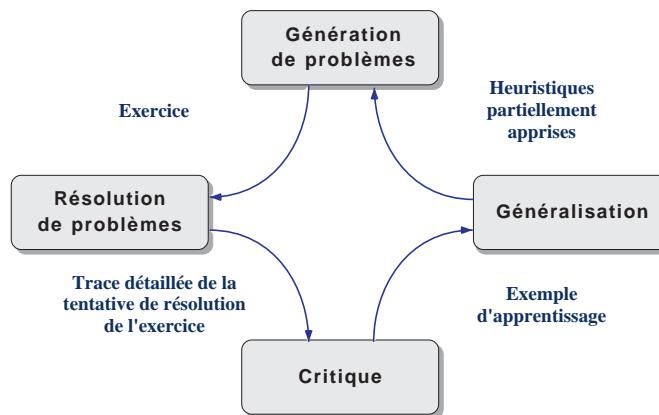


FIG. 4.20 – L'architexture générale du système LEX avec le cycle d'utilisation.

4.4 Analyse de l'algorithme d'élimination de candidats

4.4.1 Complexité au pire

L'algorithme d'élimination des candidats dépend linéairement de la taille de l'ensemble des exemples. En revanche, le nombre d'opérations nécessaires est en dépendance quadratique par rapport à la taille des ensembles frontière S et G . Du point de vue capacité mémoire, la taille nécessaire est simplement celle des ensembles frontière.

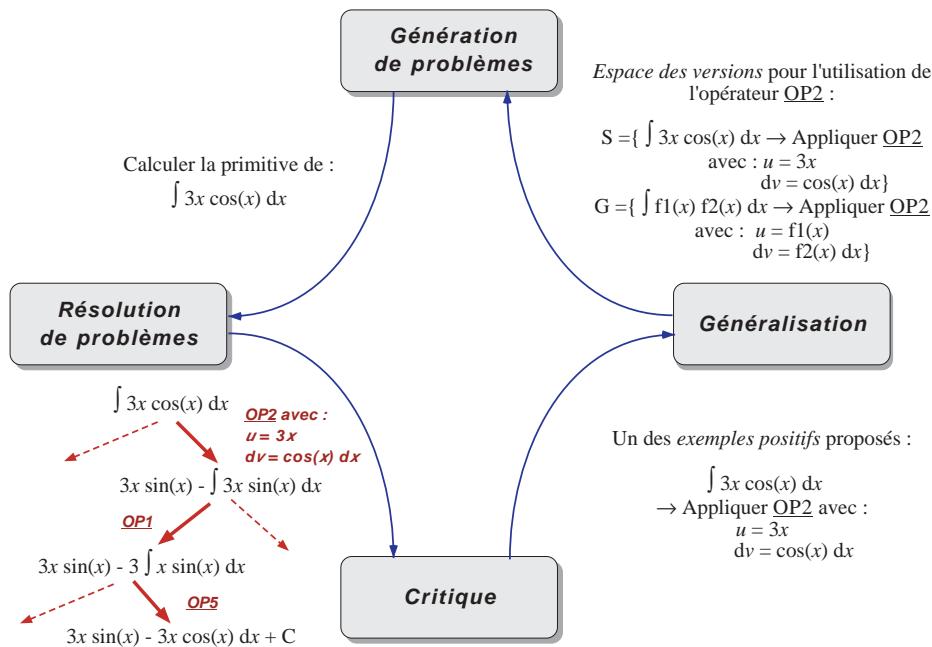


FIG. 4.21 – Un exemple de cycle d'apprentissage dans le système LEX.

La partie la plus coûteuse de l'algorithme est le test qui vérifie pour chaque couple de généralisations de l'ensemble frontière en construction l'absence de relation d'ordre entre les deux éléments. L'espoir est que le nombre des éléments de l'ensemble frontière reste petit par rapport au nombre d'exemples.

Hélas, la taille des ensembles frontière peut rapidement atteindre des valeurs importantes dans des cas réels. Il est possible de trouver un exemple simple où la taille de l'ensemble G croît exponentiellement avec le nombre de contre-exemples présentés.

Supposons que chaque exemple soit décrit par un vecteur de $2m$ attributs booléens x_i , $i = 1, \dots, 2m$ et que le langage des généralisations corresponde à l'ensemble des conjonctions qu'on puisse former sur les attributs. On se donne la présentation suivante des exemples :

1. Un exemple positif dont tous les attributs sont à *VRAI*
2. m exemples négatifs dont tous les attributs sont à *VRAI*, sauf x_j et x_{j+m} , $j = 1, \dots, m$, qui sont *FAUX*.

L'ensemble G résultant est $\{x_{i_1} \wedge x_{i_2} \dots \wedge x_{i_m} \mid i_k \in \{k, k+m\}\}$ et contient 2^m éléments.

4.4.2 Le point de vue de l'apprentissage *PAC*

On peut aussi relier les concepts introduits dans ce chapitre à ceux de l'apprentissage *PAC* (voir les chapitres 2 et 17). Si la théorie classique de l'espace des versions a pour but de décrire l'ensemble exhaustif des solutions et de les caractériser par les deux ensembles S et G , il est néanmoins possible de se demander ce que l'introduction d'une distribution de probabilité sur les exemples permet de dire sur celle des solutions.

Supposons disposer, en plus des hypothèses précédentes, d'un algorithme d'apprentissage ayant la propriété suivante : soit il produit un élément unique appartenant à l'ensemble \mathcal{H} des hypothèses qu'aurait produit l'algorithme de l'espace des versions, soit il répond que l'ensemble \mathcal{H} est incompatible avec les exemples proposés.

On peut alors démontrer la propriété suivante :

Propriété 4.2

Pour tout couple de valeurs $\epsilon, \delta \in [0, 1]$ on se donne

$$(4 \log(2/\delta) + 8VC(\mathcal{H})\log(13/\epsilon))/\epsilon$$

exemples du concept h tirés indépendamment, où $VC(\mathcal{H})$ est la VC-dimension de \mathcal{H} . Alors, avec une probabilité au moins égale à $1 - \delta$, l'algorithme d'apprentissage rendra comme résultat :

- soit une hypothèse de \mathcal{H} avec une erreur inférieure à ϵ vis à vis de h
- soit la réponse que h n'est pas dans \mathcal{H} .

Ce résultat est indépendant de la distribution de probabilité avec laquelle sont tirés les exemples.

Un cas particulier est celui où l'ensemble \mathcal{H} est de taille finie : on sait alors que sa VC-dimension l'est aussi, et on peut se ramener directement au cas présenté au chapitre 2.

Propriété 4.3

Soit \mathcal{H} de taille finie et un ensemble de m exemples d'un concept-cible h . Pour :

$$N \geq (\log(1/\delta) + \log | H |)/\epsilon$$

la probabilité d'avoir la propriété « toutes les solutions de l'espace des versions construit sur les exemples ont une probabilité d'erreur inférieure à ϵ vis-à-vis de h » est d'au moins $1 - \delta$.

Les deux propriétés précédentes quantifient donc à la manière PAC une approximation *a priori* de la qualité de l'apprentissage par l'espace des versions en fonction du nombre d'exemples présentés.

À titre d'illustration, en supposant qu'il s'agisse d'apprendre des $k - DNF$ ⁹ par cette technique, on peut remplacer dans la formule ci-dessus $\log | H |$ par $(2d)^k$. Si les exemples sont décrits par 10 attributs, que k vaut 3 et que l'on se fixe ϵ et δ à 1/10, il faudra prendre : $m \geq 100$.

4.5 La représentation des connaissances par un treillis de Galois

4.5.1 La construction de la structure

Nous nous plaçons maintenant dans une situation techniquement un peu différente de celle traitée par l'espace des versions, mais proche dans son esprit. Nous supposons que le langage de représentation est purement binaire, c'est-à-dire que chaque exemple ou contre-exemple se décrit par un ensemble de réponses *VRAI* ou *FAUX* à des tests qu'on lui pose. Dans cette représentation, un ensemble d'objets peut par exemple être le suivant. $\mathcal{S} = \{x_1, x_2, x_3, x_4, x_5\}$ est décrit par cinq attributs binaires dont la signification est la suivante :

x_1	<i>vole</i>
x_2	<i>a des plumes</i>
x_3	<i>pond des œufs</i>
x_4	<i>mammifère</i>
x_5	<i>nage sous l'eau</i>

9. Ou : *k-disjunctive normal form*, forme normale *k*-disjonctive. Un concept est de cette forme s'il est composé d'au plus *k* disjonctions \vee sur des termes composés de conjonctions \wedge entre les attributs binaires.

	x_1	x_2	x_3	x_4	x_5	commentaire
s_1	1	1	1	0	0	oie
s_2	0	0	1	1	1	ornithorynque
s_3	1	0	0	1	0	rhinolophe
s_4	1	1	1	0	0	cygne

L'idée de la représentation par treillis de Galois est de ne pas garder les exemples sous forme de matrice de $VRAI$ et $FAUX$, ou ici de 0 et de 1, mais de les transformer en une représentation ordonnée, comme sur la figure 4.22.

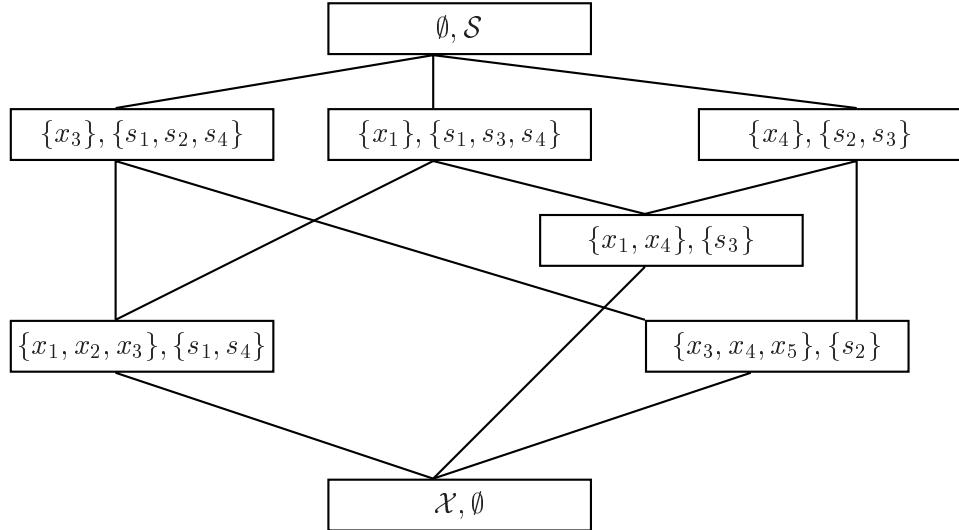


FIG. 4.22 – Un treillis de Galois.

Que signifie un tel diagramme? Le niveau supérieur de la figure, composée d'une seule case, correspond à l'absence d'attribut; le niveau en dessous, composé de trois cases, exprime la relation des exemples avec un attribut parmi les quatre; le niveau suivant exprime la relation des exemples avec deux attributs, etc. On devrait selon ce principe s'attendre à trouver dans le second niveau les cinq cases de la figure 4.23.

{ x_3 }, { s_1, s_2, s_4 } { x_1 }, { s_1, s_3, s_4 } { x_4 }, { s_2, s_3 } { x_2 }, { s_1, s_4 } { x_5 }, { s_2 }

FIG. 4.23 – La première ligne complète.

Mais on ne représente que celles qui sont indispensables: comme l'ensemble { s_1, s_4 } est strictement inclus dans l'ensemble { s_1, s_3, s_4 }, la variable x_1 rend l'écriture de x_2 inutile à ce niveau. De même, x_5 est rendue inutile par x_4 ou par x_3 .

Pour tracer un trait représentant la relation d'ordre entre les éléments de niveau différent dans le treillis de Galois, il faut que les deux composantes représentant les attributs et les exemples soient en double relation d'inclusion stricte, mais en sens inverse. Par exemple, la

case $\boxed{\{x_1\}, \{s_1, s_3, s_4\}}$ est en relation avec la case $\boxed{\{x_1, x_4\}, \{s_3\}}$ puisque $\{x_1\} \subset \{x_1, x_4\}$ et $\{s_1, s_3, s_4\} \supset \{s_3\}$.

Cette structure résume parfaitement les relations de généralité des attributs vis-à-vis des objets et, symétriquement, celles objets vis-à-vis des attributs. Il est évidemment possible de reconstituer le tableau des données d'apprentissage à partir de celle-ci. On peut aussi démontrer que le treillis de Galois construit sur une matrice binaire est unique. Divers algorithmes, soit prenant en compte tous les exemples à la fois, soit incrémentaux, ont été proposés pour réaliser cette construction. Il est important de remarquer que la taille du treillis peut être exponentielle en fonction du nombre d'attributs.

4.5.2 L'utilisation pour l'apprentissage

En quoi cette relation d'ordre particulière dans le langage des exemples peut-elle être utile à l'apprentissage ? Par la structuration qu'elle dégage des données, qui permet de les explorer de manière organisée. Si les données sont partagées en exemples positifs et négatifs et que l'on cherche à élaborer un concept correct le plus général possible, une technique est de travailler de manière ascendante dans la structure du treillis.

Sur notre exemple, supposons que les objets s_2 et s_3 soient les exemples et s_1 et s_4 les contre-exemples. La remontée dans le treillis se fait en suivant la relation d'ordre, successivement par les cases $\boxed{\mathcal{X}, \emptyset}$ $\boxed{\{x_3, x_4, x_5\}, \{s_2\}}$ $\boxed{\{x_4\}, \{s_2, s_3\}}$. Il n'est pas possible d'aller plus loin car les objets s_1 et s_4 seraient couverts. Dans ce cas précis, le concept est compatible avec les données et il est facile de vérifier que l'attribut x_4 *mammifère* forme à lui tout seul ce concept.

Dans des cas plus complexes, le test d'un concept se fait de la même façon, en contrôlant sa cohérence au fur et à mesure de la progression dans le treillis. Il est possible que celle-ci soit mise en échec si les exemples sont bruités, par exemple si l'un est à la fois négatif et positif. Dans ce cas, la structure permet de trouver le concept correct le plus général, le concept complet le moins général, ou un compromis entre les deux¹⁰.

La plupart du temps, le treillis n'est pas construit avant l'apprentissage, mais en parallèle avec la création du concept. On ne développe que la partie nécessaire au fur et à mesure. Cette technique permet d'éviter une explosion combinatoire, mais oblige à faire des choix sur lesquels on ne pourra pas toujours revenir.

Nous n'avons pas précisé quel espace d'hypothèses ni quelle méthode d'apprentissage utiliser, car tous les deux sont à la disposition de l'utilisateur : la structure en treillis de Galois induit une technique cohérente d'exploration des données quel que soit le type du concept cherché¹¹ et quel que soit l'algorithme d'apprentissage proprement dit. Par exemple, les méthodes d'apprentissage par plus proches voisins (chapitre 14) sont utilisées dans ce cadre ([NN97]).

Notes historiques et sources bibliographiques

La thèse de Tom Mitchell en 1978 : *Version spaces : an approach to concept learning* a marqué un tournant fondamental dans l'approche de l'apprentissage artificiel. Jusque là en effet, mis à part les travaux portant sur la reconnaissance des formes, l'intelligence artificielle était essentiellement guidée par l'étude de la cognition naturelle, humaine en particulier. Dans cette optique, les systèmes d'apprentissage développés représentaient des tentatives de simuler la cognition

10. Rappelons qu'un concept est *correct* s'il ne couvre aucun exemple négatif, *complet* s'il couvre tous les exemples positifs, *compatible* ou *cohérent* s'il est correct et complet.

11. Souvent écrit sous la forme d'une formule en logique des propositions.

humaine sur des tâches particulières. Ainsi, le système ARCH de P. Winston (1970) [Win70], simulait l'apprentissage d'un concept (celui d'arche) à partir d'exemples positifs et négatifs d'arches. Le système AM de Doug Lenat [Len78] simulait le raisonnement d'un mathématicien en train d'aborder la théorie des nombres et de faire des conjectures dans ce domaine. Dans tous les cas, il était supposé que le concept cible (ou la connaissance cible) était connaissable par un agent humain, et donc par le système apprenant censé le simuler. D'une part, cela allait de pair avec des approches théoriques de l'apprentissage portant sur l'*identification* exacte du concept cible et non sur une approximation. D'autre part, cela conduisait à considérer des algorithmes explorant l'espace des hypothèses possibles en adaptant et en modifiant progressivement une hypothèse unique, de même qu'apparemment un agent humain raisonne sur la base de la meilleure hypothèse courante et l'adapte si nécessaire. Les idées contenues dans la thèse de Tom Mitchell ont profondément bouleverser ce point de vue.

D'abord, l'idée d'espace des versions, l'ensemble de toutes les hypothèses cohérentes avec les données d'apprentissage, met soudain à distance les systèmes artificiels et leurs contreparties naturelles. Cela autorise à étudier des algorithmes d'apprentissage nouveaux et sans nécessairement de plausibilité psychologique. L'algorithme d'élimination des candidats en est un exemple. Ensuite, il devient naturel de s'interroger sur l'espace des hypothèses même et sur sa capacité à contenir le concept cible. Cela a conduit Mitchell à souligner l'inévitable biais pour apprendre. Comme nous l'avons déjà amplement discuté dans les chapitres 1 et 2, la possibilité de l'induction est complètement dépendante de la richesse de l'espace des hypothèses. Avant la thèse de Mitchell, les chercheurs en intelligence artificielle examinaient en quoi la représentation des connaissances choisie était ou non favorable à des raisonnements pertinents pour le domaine considéré (c'est en particulier toute l'essence des recherches de Lenat), en revanche il n'était pas question de s'interroger sur la possibilité, encore moins la nécessité, d'avoir un espace d'hypothèses limité. La réalisation progressive de ce dernier point s'accompagne de l'essor des travaux portant sur des théories de l'apprentissage comme techniques d'approximation et non plus comme identification d'un concept cible. Le développement concomitant du connexionnisme dans les années quatre-vingt joue alors un rôle de catalyseur en permettant l'intrusion des mathématiques du continu, et donc des outils de l'analyse mathématique, comme l'optimisation et la convergence, dans l'étude de l'apprentissage.

Pour terminer par une note philosophique, il est remarquable que la vision de l'apprentissage comme sélection de bonnes hypothèses au sein d'un ensemble d'hypothèses possibles donné *a priori* s'accorde à la vision actuelle de la biologie. La théorie de l'évolution de Darwin, celle de Changeux et de ses collègues (qui voient l'apprentissage comme élimination de connexions dans le cerveau), et la théorie de Chomsky sur l'apprentissage de la langue naturelle comme spécialisation d'une grammaire universelle définissant l'enveloppe des langues possibles, toutes ces démarches théoriques vont à l'unisson. L'approche actuelle de l'apprentissage artificiel, considérant l'apprentissage comme la sélection des hypothèses les plus performantes par rapport aux observations, s'est finalement jointe à ce mouvement. L'avenir nous dira la destinée de cet étonnant exemple multidisciplinaire de pensée unique.

Nous donnons dans ce qui suit quelques indications bibliographiques aux lecteurs intéressés par les recherches et les développements portant sur les espaces des versions.

Les origines de l'apprentissage par généralisation et spécialisation remontent au moins à P. Winston ([Win75]) ; la formalisation et la résolution du problème par l'espace des versions sont une des étapes majeures de la naissance de la discipline. L'essentiel de la théorie et de l'algorithme a été produit par T. Mitchell [Mit82]. La présentation qui en est faite ici et l'exemple des rectangles sont repris du texte de Nicolas dans [Nic93]. Les concepts introduits et l'algorithme d'élimination des candidats peuvent être trouvés dans presque tous les livres

d'intelligence artificielle. Nous recommandons particulièrement la présentation récente de T. Mitchell [Mit97].

L'algorithme d'élimination des candidats a fait l'objet de l'examen critique du théoricien David Haussler [Hau88] qui a souligné en particulier que la taille de la borne G de l'espace des versions pouvait croître exponentiellement avec le nombre d'exemples négatifs. Cette observation relativise évidemment l'avantage apporté par la considération de ces bornes. Cependant, l'examen de la preuve de Haussler laisse supposer que le phénomène de croissance exponentielle ne peut se produire que pour des échantillons de données très particuliers, et présentés dans un ordre très défavorable. Des chercheurs comme Hirsh [Hir90] ou Smith et Rosenbloom [SR90] ont proposé des heuristiques pour améliorer l'ordre de présentation des exemples. Une limite plus sérieuse de l'approche de Tom Mitchell concerne les données bruitées, c'est-à-dire mal décrites ou mal classées. L'insistance sur la stricte cohérence des hypothèses de l'espace des versions avec les exemples condamne généralement l'algorithme original à ne pas pouvoir trouver d'hypothèse s'accordant aux données. Des propositions ont donc été faites visant à relâcher l'exigence de stricte cohérence. Hirsh [Hir90, Hir92] a ainsi présenté un algorithme d'apprentissage dans lequel on forme un espace des versions pour chaque exemple positif, puis on en fait l'intersection. Il montre que cette idée permet de traiter des données bruitées. Michèle Sebag [Seb94a, Seb94b] a poussé cette idée encore plus loin avec l'approche de *disjunctive version spaces* qui marie une approche de généralisation en représentation attribut-valeur avec la technique de l'espace des versions.

L'apprentissage grâce à la structure en treillis de Galois a été en particulier étudié par [Gan93, Wil92a], [LS98]. La théorie de ces espaces est développée dans [Bir67]. L'apprentissage par plus proche voisins en liaison avec la construction de cette structure est en particulier étudiée dans [NN97].

Résumé

- La méthode de l'espace des versions vise à définir tous les concepts cohérents avec un ensemble d'exemples.
- Comme le nombre de ces concepts peut être infini, on s'intéresse à une définition de leur ensemble en intension.
- Celle-ci est définie par deux ensembles finis S et G et une relation d'ordre sur les concepts.
- La recherche d'un concept particulier se fait, comme dans les treillis de Galois, en exploitant la structure algébrique des solutions potentielles.
- La méthode de l'espace des versions est d'une grande importance historique et méthodologique. Son intérêt pratique est loin d'être nul.

Chapitre 5

La programmation logique inductive

La programmation logique inductive (PLI) réalise l'apprentissage de formules de la logique des prédicats à partir d'exemples et de contre-exemples. L'enjeu est de construire des expressions logiques comportant des variables liées les unes aux autres. Par exemple, à partir de la description des liens de parenté dans quelques familles (« Jean est le père de Pierre », « Paul est le père de Jean », « Paul est le grand-père de Pierre »...), un programme de PLI doit être capable de trouver une formule du type « Pour tous les x et z tels que z est le grand-père de y , il existe x tel que x est le père de y et y est le père de z ».

Ce type d'apprentissage est difficile à réaliser. On se limite la plupart du temps à un sous-ensemble de la logique des prédicats qu'on appelle « programme logique », en référence aux langages de programmation du type Prolog qui travaillent directement dans ce langage. La PLI a deux caractéristiques fortes : d'abord, le langage de représentation des hypothèses est très bien connu mathématiquement et algorithmiquement. La notion de généralisation peut donc être introduite en cohérence avec les outils de la logique, comme la démonstration automatique. Ensuite, du fait de la richesse de ce langage, la combinatoire de l'apprentissage est très grande : il s'agit d'explorer un espace immense en faisant constamment des choix qu'il sera difficile de remettre en question. C'est pourquoi il est important en PLI de bien décrire les biais d'apprentissage qui limitent cette exploration.

Comme le langage de description des concepts est riche, la PLI peut s'appliquer à un grand nombre de situations. En pratique, les algorithmes permettent d'apprendre des concepts opératoires dans des domaines aussi variés que le traitement de la langue naturelle, la chimie, le dessin industriel, la fouille de données, etc.

L'ORNITHOLOGIE s'apprend vite : le débutant que nous avons rencontré dans l'avant-propos de ce livre sait maintenant reconnaître nombre d'espèces : des rapaces, comme l'épervier et le faucon pèlerin ou des palmipèdes, comme la sarcelle et la bernache. Il rencontre à nouveau l'expert, qui se montre satisfait de ses progrès (mais conscient de ses limites) et qui lui propose un nouvel exercice : apprendre à reconnaître le sexe de l'animal (sans dissection) sur des espèces où seule la taille peut éventuellement aider. L'ensemble d'apprentissage proposé par l'expert donne pour mesure l'envergure en centimètres. Il est le suivant :

	espèce	envergure	sexé
\mathbf{x}_1	épervier	60	mâle
\mathbf{x}_2	épervier	80	femelle
\mathbf{x}_3	pèlerin	90	mâle
\mathbf{x}_4	pèlerin	110	femelle
\mathbf{x}_5	sarcelle	70	mâle
\mathbf{x}_6	sarcelle	70	femelle

Le débutant s'aperçoit rapidement qu'il ne sait pas trouver un concept pour distinguer les mâles des femelles. C'est très simple à voir : les exemples \mathbf{x}_5 et \mathbf{x}_6 sont décrits par les mêmes attributs et sont associés à des étiquettes contradictoires. Il fait part de cette réflexion à l'expert. « D'accord », lui répond ce dernier, « je sais que vous connaissez la méthode de l'espace des versions et que vous essayez de trouver un concept écrit dans un certain langage des hypothèses. Vous avez choisi pour ce langage la logique des propositions sur des sélecteurs portant sur des attributs. N'est-ce pas ? C'est en effet impossible d'apprendre quoi que ce soit dans ce langage sur ces données. Je vais vous initier à une autre façon de faire de l'apprentissage. »

L'expert propose alors de reformuler les données sous la forme de prédicats, ce qui consiste par exemple à écrire l'attribut « envergure » pour l'exemple \mathbf{x}_5 de la manière suivante :

$$\text{envergure}(\mathbf{x}_5, 90) = \text{VRAI}$$

Dans ce formalisme, le premier exemple devient :

$$(\text{epervier}(\mathbf{x}_1) = \text{VRAI}) \wedge (\text{envergure}(\mathbf{x}_5, 60) = \text{VRAI}) \wedge (\text{male}(\mathbf{x}_1) = \text{VRAI})$$

ou plus simplement :

$$\text{epervier}(\mathbf{x}_1) \wedge \text{envergure}(\mathbf{x}_5, 60) \wedge \text{male}(\mathbf{x}_1)$$

L'ensemble des exemples est maintenant l'union (le *OU* ou \vee) d'expressions de ce type, c'est-à-dire une formule logique formée de la disjonction des exemples, chacun écrit comme une conjonction *ET* (\wedge) de prédicats sur ses attributs et sa supervision.

« Et alors ? Est-ce que la contradiction entre les exemples \mathbf{x}_5 et \mathbf{x}_6 a disparu pour autant ? » demande le débutant. « Non », répond l'expert, « vous avez raison, ce n'est qu'une réécriture pour l'instant. Mais je vais vous donner un peu de matériel supplémentaire. »

Il propose alors d'utiliser deux nouveaux prédicats $\text{env-sup}(\mathbf{x}, \mathbf{y})$ et $\text{meme-esp}(\mathbf{x}, \mathbf{y})$ avec la signification suivante : $\text{env-sup}(\mathbf{x}, \mathbf{y})$ est *VRAI* quand l'envergure de l'exemple noté \mathbf{x} est strictement supérieure à celle de l'exemple noté \mathbf{y} et $\text{meme-esp}(\mathbf{x}, \mathbf{y})$ est *VRAI* si l'exemple \mathbf{x} est de la même espèce que l'exemple \mathbf{y} . Les exemples deviennent alors, en notant pour raccourcir :

$\text{epervier}(\mathbf{x})$ par $E(\mathbf{x})$

$\text{pelerin}(\mathbf{x})$ par $P(\mathbf{x})$

sarcelle(x) par $S(x)$

envergure(x, 60) par $e(x, 60)$, etc.

	$E(x)$	$P(x)$	$S(x)$	$B(x)$	$e(x, 60)$	$e(x, 70)$
x_1	VRAI	FAUX	FAUX	FAUX	VRAI	FAUX
x_2	VRAI	FAUX	FAUX	FAUX	FAUX	FAUX
x_3	FAUX	VRAI	FAUX	FAUX	FAUX	FAUX
x_4	FAUX	VRAI	FAUX	FAUX	FAUX	FAUX
x_5	FAUX	FAUX	VRAI	FAUX	FAUX	VRAI
x_6	FAUX	FAUX	VRAI	FAUX	FAUX	VRAI

	$e(x, 80)$	$e(x, 90)$	$e(x, 100)$	$e(x, 110)$	$male(x)$	$fem(x)$
x_1	FAUX	FAUX	FAUX	FAUX	VRAI	FAUX
x_2	VRAI	FAUX	FAUX	FAUX	FAUX	VRAI
x_3	FAUX	VRAI	FAUX	FAUX	VRAI	FAUX
x_4	FAUX	FAUX	VRAI	FAUX	FAUX	VRAI
x_5	FAUX	FAUX	FAUX	FAUX	VRAI	FAUX
x_6	FAUX	FAUX	FAUX	FAUX	FAUX	VRAI

Il faut écrire aussi la table de vérité des prédicts relationnels *env-sup* et *meme-esp*. La seconde est évidente et la première débute ainsi :

	x_1	x_2	x_3	x_4	x_5	x_6
x_1	FAUX	FAUX	FAUX	FAUX	FAUX	FAUX
x_2	VRAI	FAUX	FAUX	FAUX	VRAI	VRAI
x_3	VRAI	VRAI	FAUX	FAUX	VRAI	VRAI
...

Quoiqu'un peu impressionné par la quantité plus importante de données, le débutant tente l'apprentissage d'un concept dans le même langage. Il obtient par exemple :

$$\begin{aligned} male &= male(x_1) \vee male(x_3) \vee male(x_5) \\ &= [E(x_1) \wedge e(x_1, 60)] \vee [P(x_3) \wedge e(x_3, 90)] \vee [S(x_5) \wedge e(x_5, 70)] \end{aligned}$$

Mais cela ne le satisfait pas, car il n'y a aucune généralisation. Il essaie donc d'utiliser la connaissance supplémentaire fournie par l'expert et obtient ceci, après avoir essayé un grand nombre de formules :

$$\begin{aligned} &env\text{-}sup(x_2, x_1) \wedge E(x_1) \wedge E(x_2) \wedge male(x_1) \wedge fem(x_2) \\ &\wedge env\text{-}sup(x_4, x_3) \wedge P(x_3) \wedge P(x_4) \wedge male(x_3) \wedge fem(x_4) \\ &\wedge meme\text{-}esp(x_1, x_2) \wedge meme\text{-}esp(x_3, x_4) \end{aligned}$$

« Bon, dit l'expert, nous y sommes presque. Nous savons tous les deux que les éperviers et les faucons sont des rapaces, mais pas les sarcelles. Pouvez-vous utiliser cette information ? »
 « Voyons cela », répond le débutant :

$$\begin{aligned} &env\text{-}sup(x_2, x_1) \wedge rapace(x_1) \wedge rapace(x_2) \wedge male(x_1) \wedge fem(x_2) \\ &\wedge env\text{-}sup(x_4, x_3) \wedge rapace(x_3) \wedge rapace(x_4) \wedge male(x_3) \wedge fem(x_4) \\ &\wedge meme\text{-}esp(x_1, x_2) \wedge meme\text{-}esp(x_3, x_4) \end{aligned}$$

« Cette fois, c'est presque bon. Je vous donne une dernière clé. Que dites-vous de la formule suivante? »

$$\text{env-sup}(\mathbf{X}, \mathbf{Y}) \wedge \text{rapace}(\mathbf{X}) \wedge \text{rapace}(\mathbf{Y}) \wedge \text{meme-esp}(\mathbf{X}, \mathbf{Y}) \wedge \text{male}(\mathbf{Y}) \wedge \text{fem}(\mathbf{X})$$

« Elle signifie que, pour toutes les espèces de rapaces (et non pour les autres oiseaux), la femelle d'une espèce est d'envergure supérieure au mâle » répond le débutant « mais je n'aurais pas su la trouver : je ne connaissais ni les prédictats relationnels ni les expressions avec des variables ».

Notations utiles pour le chapitre

\models	Relation d'implication sémantique (théorie des modèles)
\vdash	Relation d'implication logique (théorie de la preuve)
\models_T	Relation d'implication relative à la théorie T
\wedge	ET : conjonction
\vee	OU : disjonction
\neg	Négation
$h_1 \succcurlyeq h_2$	L'hypothèse h_1 est plus générale que h_2

5.1 La programmation logique inductive : le cadre général

5.1.1 Complexité de l'induction et expressivité du langage d'hypothèses

Le chapitre 4 a montré comment on peut formaliser le problème de l'induction de concept, c'est-à-dire de l'apprentissage d'une fonction indicatrice, à valeur dans $\{0, 1\}$. L'idée essentielle est de considérer l'ensemble de toutes les hypothèses cohérentes¹ avec l'échantillon d'apprentissage $\mathcal{S} = \{(\mathbf{x}_1, u_1), (\mathbf{x}_2, u_2), \dots, (\mathbf{x}_m, u_m)\}$. Cet espace est appelé l'espace des versions. Sa mise à jour se fait incrémentalement à chaque fois qu'un nouvel exemple d'apprentissage devient disponible. Elle passe par la détermination de la plus petite généralisation (respectivement spécialisation) d'une hypothèse et du nouvel exemple lors de l'adaptation du S-set (respectivement du G-set). Nous avons vu dans le chapitre 4 que les concepts de généralisation et de spécialisation se définissent par référence à la relation d'inclusion entre les *extensions* des concepts, c'est-à-dire l'ensemble des objets de \mathcal{X} qu'ils couvrent. Un concept est dit plus spécifique qu'un autre si son extension est incluse dans l'extension de l'autre. On obtient ainsi une relation d'ordre partiel qui est exploitée dans l'algorithme d'élimination des candidats pour mettre à jour les deux bornes de l'espace des versions : le S-set et le G-set.

Cette approche de l'apprentissage soulève deux problèmes. Le premier est que la notion de couverture d'un exemple par une hypothèse n'est pas toujours aussi simple que le chapitre 4 peut le laisser penser. Nous revenons sur ce problème dans la section suivante. Le deuxième problème est que les régularités observées sur les parties de \mathcal{X} ne se transportent pas complètement dans l'espace des concepts \mathcal{H} défini par le langage $\mathcal{L}_{\mathcal{H}}$ d'expression des hypothèses. Ainsi, l'ensemble des parties de \mathcal{X} forme une algèbre, ce qui signifie qu'il existe des opérations bien définies

1. C'est-à-dire, rappelons-le, couvrant tous les exemples positifs et excluant tous les exemples négatifs de cet échantillon.

pour calculer le plus petit ensemble d'exemples contenant deux ensembles d'exemples, de même que pour calculer le plus grand ensemble d'exemples contenus à la fois dans deux ensembles d'exemples. Ce sont les opérations classiques d'union et d'intersection. Malheureusement, les opérations correspondantes sur l'espace des concepts \mathcal{H} : la *plus petite généralisation* (*least general generalization, lgg*) et la *spécialisation maximale* (*most general specialization, mgs*) ne sont pas en général définies de manière unique. Cela provient du fait que tout ensemble d'exemples ne correspond pas forcément à un concept, de même que toute expression dans le langage des hypothèses $\mathcal{L}_{\mathcal{H}}$ n'a pas nécessairement une contrepartie dans \mathcal{X} . Nous avons en effet vu la nécessité de l'existence d'un biais de langage, limitant la richesse de \mathcal{H} . De ce fait, la complexité de l'apprentissage de concept dépend fortement de $\mathcal{L}_{\mathcal{H}}$, le langage d'expression des hypothèses.

En fonction du langage $\mathcal{L}_{\mathcal{H}}$, il peut être possible, ou impossible, de définir une relation $\succcurlyeq \subseteq \mathcal{L}_{\mathcal{H}} \times \mathcal{L}_{\mathcal{H}}$ de généralité intensionnelle (dans \mathcal{H}), appelée *subsomption*, qui coïncide avec la relation d'inclusion dans \mathcal{X} . Si $h_1 \succcurlyeq h_2$ implique que h_1 est aussi générale que h_2 , la relation de subsomption est dite *saine* (*sound*) ; si h_1 est aussi générale que h_2 implique que $h_1 \succcurlyeq h_2$, alors la relation de subsomption est dite *complète* (*complete*). Lorsque la relation de subsomption est saine, on peut démontrer que l'espace des versions correspondant à un ensemble d'exemples est convexe par rapport à la relation \succcurlyeq . Grâce à cela, on peut alors représenter l'espace des versions par une borne inférieure : le S-set, et par une borne supérieure : le G-set.

La complexité de l'induction supervisée dépend donc du langage d'expression des hypothèses \mathcal{H} . Dans le chapitre 4, nous avons essentiellement fait référence à des langages d'hypothèses en attributs-valeurs. Ceux-ci sont souvent insuffisants pour décrire des domaines dans lesquels il est nécessaire de pouvoir décrire des relations (comme dans le domaine des arches décrits dans le chapitre 2 ou dans l'exemple introductif avec les relations de comparaison d'envergure). C'est pourquoi on est tenté d'utiliser la logique des prédictats ou logique du premier ordre. Mais est-il alors raisonnable de vouloir pratiquer de l'induction avec un tel langage d'expression des hypothèses ? Nous examinons le prix à payer dans la suite.

5.1.2 La relation de couverture en logique du premier ordre

L'objectif de la programmation logique inductive (PLI) est de construire des programmes logiques à partir d'exemples supervisés. Schématiquement, un programme logique est un ensemble de règles de la forme *prémisses* \rightarrow *conclusion*. Selon que l'on autorise l'usage de la négation pour formuler les prémisses ou non, on parle de programmes normaux ou de programmes définis. Étant donné leur avantage en terme de pouvoir expressif, nous étudierons dans ce chapitre l'apprentissage des programmes normaux. Par ailleurs, deux grandes familles d'approches sont considérées en programmation logique inductive :

1. *L'apprentissage empirique* dans lequel on dispose de nombreux exemples et contre-exemples pour apprendre un nouveau concept (i.e. un programme). On cherche alors à apprendre une définition qui permette d'expliquer (couvrir) tous les exemples positifs connus mais aucun contre-exemple.
2. *L'apprentissage interactif*, dans lequel on cherche à adapter une description (ou théorie) du domaine en fonction de quelques nouveaux exemples et contre-exemples. On parle aussi dans ce cas de *révision de connaissances*.

Le deuxième type d'apprentissage implique en général des raisonnements et des mécanismes de généralisation beaucoup plus complexes, et malheureusement moins maîtrisés que dans le premier. Pour des raisons de place, nous nous limitons donc dans ce chapitre à l'étude de l'apprentissage empirique.

Dans le langage de description des hypothèses par attributs-valeurs, une expression peut prendre la forme :

$$(taille = grande) \vee (couleur = rouge) \vee (forme = carré)$$

dénarrant le concept « grands carrés rouges ». La même expression sert à la fois à dénoter un exemple (un certain « grand carré rouge ») et un ensemble d'exemples (tous les grands carrés rouges). La logique par attributs-valeurs n'est pas capable de faire la distinction entre les exemples et les hypothèses. Cela révèle un manque de pouvoir expressif, mais permet l'astuce de la représentation unique (*single representation trick*), ce qui signifie en pratique que le test de couverture d'un exemple par une hypothèse est le même que le test de subsomption entre deux hypothèses. Ce test est aisément dans le cas de la logique des attributs-valeurs et de la logique des propositions en général.

Prenons le cas d'expressions conjonctives en logique attributs-valeurs. Le test de subsomption entre deux expressions revient à contrôler que chaque paire attribut-valeur qui apparaît dans l'expression la moins générale apparaît aussi dans l'autre. Le processus de généralisation d'une expression consiste à monter dans la hiérarchie définie par la relation de subsomption. La généralisation revient donc à abandonner un terme de la conjonction. Par exemple, si le concept :

$$(taille = moyenne) \vee (couleur = rouge) \vee (forme = cercle)$$

doit être généralisé pour couvrir l'exemple :

$$(taille = petite) \vee (couleur = rouge) \vee (forme = cercle) \vee (poids = lourd)$$

il suffit de laisser tomber $(taille = moyenne)$ pour obtenir le concept adéquat :

$$(couleur = rouge) \vee (forme = cercle)$$

Il est à noter que la généralisation obtenue est minimale (*lgg*).

De la même manière, spécialiser une expression revient à ajouter une paire attribut-valeur dans la conjonction. Par exemple, si le concept :

$$(couleur = rouge) \vee (forme = cercle)$$

ne doit pas couvrir l'exemple négatif :

$$(taille = grande) \vee (couleur = rouge) \vee (forme = cercle)$$

il suffit d'ajouter au concept une paire d'attributs-valeurs non présente dans l'exemple négatif pour le spécialiser assez. Il faut noter qu'en revanche il n'y a pas ici de spécialisation maximale unique.

En logique des prédictats, il est nécessaire de reconsidérer les notions de couverture, de subsomption, et par voie de conséquence les opérations de généralisation et de spécialisation.

Considérons à nouveau des concepts conjonctifs, mais cette fois exprimés en logique du premier ordre. Par exemple, en utilisant la syntaxe des programmes Prolog², le `concept1` à apprendre pourrait se représenter³ par :

2. Dans cette syntaxe, le symbole `:-` signifie l'implication du membre gauche par le membre droit et la virgule dans le membre droit est la conjonction.

3. Soit, en notation logique traditionnelle : `rouge(X) ∧ cercle(X) → concept1(X)`.

```
concept1(X) :- rouge(X), cercle(X).
```

Un concept peut être défini par plusieurs clauses, par exemple :

```
concept2(X) :- rouge(X), carré(X).
```

```
concept2(X) :- vert(X), cercle(X).
```

ce qui signifie que le concept correspond aux objets qui sont soit des carrés rouges, soit des cercles verts.

Les exemples sont décrits par des atomes clos (sans variable), par exemple :

```
petit(petitcerclerouge).
```

```
rouge(petitcerclerouge).
```

```
cercle(petitcerclerouge).
```

Ici, `petitcerclerouge` est un exemple de `concept1` car `concept1(petitcerclerouge)` peut être prouvé. D'une manière générale, étant donnée une conjonction d'atomes clos `Desc(Exemple)` décrivant `Exemple`, un concept `Concept(X) :- Conditions(X)` classe `Exemple` positivement si :

$$T \wedge \text{Desc}(\text{Exemple}) \wedge (\text{Concept}(X) \text{ :- } \text{Conditions}(X)) \models \text{Concept}(\text{Exemple}).$$

On suppose donc de manière générale que l'on dispose d'une connaissance initiale T , ou *théorie du domaine*, mise sous forme de programme logique. Pour tester si un exemple est couvert par un concept, on ajoute sa description dans la théorie du domaine, et on chercher à prouver⁴ `Concept(Exemple)` à l'aide de la définition du concept. Si la preuve échoue, on interprète cet échec comme une classification négative (négation par l'échec). Dans ce cas, il faudra modifier la définition du concept. Pour cela il faut examiner la notion de subsomption en logique du premier ordre.

5.1.3 La subsomption en logique du premier ordre

Il s'agit maintenant de définir la relation de généralité ou subsomption entre clauses. Par exemple, on s'attend à ce que la clause :

`concept3(X) :- rouge(X), cercle(X).` subsume, donc soit plus générale, que la clause :

```
concept3(X) :- petit(X), rouge(X), carré(X).
```

En effet, on constate⁵ que l'extension correspondant à la première définition inclut l'extension correspondant à la seconde définition. Il se trouve d'ailleurs que la seconde clause contient les mêmes littéraux que la première et qu'elle est donc, selon notre définition de la section précédente, plus spécifique. Cependant, ce cas n'épuise pas la notion de subsomption de deux clauses. Prenons par exemple les deux clauses suivantes :

```
concept4(X) :- carré(X), triangle(Y), mêmecouleur(X,Y).
```

```
concept4(X) :- carré(X), triangle(t), mêmecouleur(X,t).
```

La première clause décrit l'ensemble des carrés de même couleur que les triangles existants. La seconde clause décrit l'ensemble des carrés ayant la même couleur qu'un triangle particulier t . Il est évident que le deuxième ensemble est inclus dans le premier et donc que la première clause subsume la seconde. Il faut donc rendre compte aussi de ce cas de subsomption.

5.1.3.1 La θ -subsomption

En combinant les deux cas précédents, on arrive ce qu'on appelle la θ -subsomption, qui se définit informellement ainsi :

-
- 4. Le symbole \models correspond à la notion sémantique de l'implication. Nous reviendrons sur ce formalisme dans la section suivante.
 - 5. Nous restons volontairement informels dans cette première exposition des notions de couverture et de subsomption. Les paragraphes suivants seront rigoureux.

`Clause1` subsume `Clause2` s'il existe une substitution θ applicable à `Clause1` et telle que tous les littéraux dans la clause ainsi obtenue apparaissent dans `Clause2`.

Il est à noter que si `Clause1` θ -subsume `Clause2`, alors on a aussi `Clause1` \models `Clause2`. En revanche, la réciproque n'est pas toujours vraie, comme le montre l'exemple suivant :

```
list([V|W]) :- list(W).
list([X,Y|Z]) :- list(Z).
```

Étant donnée la liste vide, la première clause construit des listes de n'importe quelle longueur, tandis que la seconde construit des listes de longueur paire. Toutes les listes construites par la seconde clause peuvent aussi l'être par la première, qui est donc plus générale. Pourtant, il n'y a pas de substitution applicable à la première clause et permettant d'obtenir la seconde (une telle substitution devrait appliquer `W` à la fois sur `[Y|Z]` et sur `Z`, ce qui est impossible). La θ -subsumption est donc plus faible que l'implication. Elle a en outre des limitations rédhibitoires si l'on veut induire des clauses récursives. Soit en effet, les clauses : `p(f(f(a))) :- p(a)` et `p(f(b)) :- P(b)`. Si l'on cherche la plus petite généralisation par rapport à la θ -subsumption, on trouve la clause : `p(f(Y)) :- p(X)`, tandis que la clause `p(f(X)) :- p(X)`, plus satisfaisante, ne peut être trouvée. Le problème est que la θ -subsumption ne peut prendre en compte les clauses qui peuvent être résolues avec elles-mêmes.

5.1.3.2 L'implication

On pourrait envisager d'utiliser l'implication pour définir la subsumption entre clauses :

$$\text{Clause1 subsume Clause2} \quad \text{si} \quad \text{Clause1} \models \text{Clause2}$$

Cela introduit cependant deux problèmes. Le premier est qu'il s'agit d'une définition sémantique (s'appuyant sur la théorie des modèles) et qu'il reste donc à préciser la procédure effective de preuve de subsumption ainsi que la procédure permettant de généraliser une clause. Le second problème est que la plus petite généralisation (*lgg*) n'est pas toujours unique si la subsumption est définie comme l'implication logique. Soit par exemple les deux clauses :

```
list([A,B|C]) :- list(C).
list([P,Q,R|S]) :- list(S).
```

Selon l'implication logique, ces clauses ont deux *lgg*:

```
list([X|Y]) :- list(Y) et list([X,Y|Z]) :- list(V).
```

Selon la θ -subsumption, seule cette dernière est une *lgg*. Il est à noter que la première *lgg* est en réalité plus plausible.

5.1.3.3 La subsumption des théories

Jusque-là nous avons seulement considéré la subsumption entre deux clauses. Dans la plupart des cas intéressants cependant, nous devons prendre en compte des ensembles de clauses décrivant des théories⁶ sur le monde. Il faut donc définir aussi la subsumption entre théories. Par exemple, soit la théorie :

```
concept5(X) :- petit(X), triangle(X).
polygon(X) :- triangle(X).
```

Elle est impliquée logiquement par la théorie suivante :

```
concept5(X) :- polygone(X).
polygon(X) :- triangle(X).
```

puisque tout modèle de la seconde est un modèle de la première théorie. Pourtant, la clause :

6. En programmation logique, une théorie est simplement définie comme un ensemble de clauses.

```
concept5(X) :- petit(X), triangle(X).
```

n'est pas logiquement impliquée par la clause :

```
concept5(X) :- polygon(X).
```

La subsomption entre théories ne peut donc pas être réduite à la subsomption entre clauses.

5.1.3.4 La subsomption relative à une théorie

Nous devons avoir recours dans ce cas à la notion de *subsumption relative à une théorie* entre deux clauses. Par définition : Clause1 subsume Clause2 relativement à la théorie T si $T \wedge \text{Clause1} \models \text{Clause2}$, ce que nous notons : $\text{Clause1} \models_T \text{Clause2}$.

Par exemple, supposons que T contienne la clause :

```
polygon(X) :- triangle(X).
```

Nous avons alors :

```
 $T \wedge \text{concept5}(X) :- \text{polygon}(X) \models \text{concept5}(X) :- \text{petit}(X), \text{triangle}(X)$ .
```

De cette manière, nous obtenons en effet que :

```
concept5(X) :- polygon(X)
```

subsume :

```
concept5(X) :- petit(X), triangle(X)
```

relativement à T .

5.1.4 Un résumé des relations de subsomption possibles

Pour résumer, les trois types de relations de subsomption utilisés en programmation logique inductive sont la θ -subsumption, l'implication logique et la subsomption relative à une théorie du domaine. De ces trois types de subsomptions, la première est la plus aisée à réaliser (et elle est déjà NP-complète!). En particulier, la θ -subsumption est décidable, tandis que l'implication logique ne l'est pas⁷, même pour des clauses de Horn. De même, la subsomption relative est plus sévère que l'implication : les deux sont indécidables, mais les procédures de preuve pour l'implication ne nécessitent pas la prise en compte de la théorie T , contrairement aux procédures pour la subsomption relative qui doivent prendre en compte toutes les dérivations possibles à partir de $T \wedge \text{Clause}$.

En pratique, lors de l'apprentissage à partir d'exemples représentés par des clauses, en présence ou non d'une théorie du domaine, il est essentiel de pouvoir déterminer la plus petite généralisation (relative dans le cas de la subsomption relative) ainsi que leur spécialisation maximale. L'existence de ces généralisations ou spécialisations dépend à la fois du type de subsomption considéré et du langage utilisé : logique des clauses ou logique réduite aux clauses de Horn. Le tableau suivant fournit un résumé des résultats connus sur les six cas possibles (« + » correspondant à une réponse positive et « - » à une réponse négative).

Type de subsomption	Clauses de Horn		Clauses générales	
	lgg	mgs	lgg	mgs
θ -subsumption	+	+	+	+
Implication (\models)	-	-	+ si sans fonction	+
Implication relative (\models_T)	-	-	-	+

7. Church en 1932 a montré qu'il ne peut exister d'algorithme pouvant décider en un temps fini si une inférence est logiquement valide ou non en logique des prédictats standard.

Ce tableau illustre une fois de plus le compromis existant entre l'expressivité d'un langage et les raisonnements qui peuvent être réalisés. Il est par ailleurs important de noter que l'utilisation d'un langage causal sans symbole de fonction conduit à une structure de treillis. C'est pourquoi la plupart des travaux en PLI se placent dans ce cadre.

Avant d'aborder les moyens effectifs d'exploration de l'espace des hypothèses, il est nécessaire de rappeler plus formellement quelques concepts de base en logique.

5.2 La logique des prédictats et les programmes logiques : terminologie

Ce paragraphe définit de manière plus rigoureuse ce qu'est une formule de la logique des prédictats, autrement dit quels sont les concepts que nous cherchons à apprendre. Ces concepts sont construits à partir de symboles primitifs (les variables, les connecteurs, les quantificateurs, les prédictats, les fonctions et les parenthèses), en respectant une syntaxe stricte. Cette syntaxe permet, par l'application de certaines règles, de réaliser des démonstrations dans ce système formel, c'est-à-dire de déduire des théorèmes à partir d'axiomes. Finalement, une sémantique doit être proposée pour permettre une interprétation hors de ce système formel qui n'a pas de signification en soi ([NS93, GN88, Tau94]).

5.2.1 La syntaxe de la logique des prédictats

Les formules logiques sont écrites dans un langage construit à partir des *symboles primitifs* suivants :

Variables $X, Y \dots$ Une variable prend ses valeurs sur un *domaine*. Des exemples de domaines sont : l'ensemble des nombres entiers, l'ensemble des clients d'une compagnie d'assurances.

Constante $a, b, \dots, jerome, laure, \dots, 1, 2, \dots, VRAI, FAUX, \dots$ Un ensemble de constantes forme le domaine d'une variable. Une *instanciation* d'une variable est une constante de son domaine.

Connecteur $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$

Quantificateur \forall, \exists

Prédicats $P, Q, \dots, sont - mariés, \dots$

Un prédictat possède une *arité* a (le nombre d'arguments sur lequel il porte) qui doit valoir au moins 1. On note, si besoin est, le prédictat de manière plus complète par : P/a . Un prédictat est une *relation* entre plusieurs domaines de variables, autrement dit une application de l'ensemble de ces domaines dans $\{VRAI, FAUX\}$. Par exemple, pour le prédictat $sont - mariés/2$: $sont - mariés(jerome, laure) = VRAI$.

Fonction $f, g, \dots, age - ainé \dots$

Une fonction diffère d'un prédictat par la nature de son résultat, qui peut appartenir à n'importe quel domaine. Par exemple, pour la fonction $age - ainé/2$: $age - ainé(jerome, X)$ a pour valeurs l'âge de l'ainé(e) de *jerome* (leur mère n'est pas ici précisée). Une fonction d'arité 0 n'est autre qu'une constante.

Parenthèses $(,)$.

5.2.1.1 Le langage de la logique des prédictats

Terme Un *terme* est défini récursivement comme

- soit une constante ;

- soit une variable;
- soit une fonction appliquée à des termes, c'est-à-dire une expression de la forme : $f(t_1, \dots, t_m)$ où f est une fonction d'arité m et les t_i sont des termes.

Un exemple de terme: *sont – maries(pere(laure), mauricette)*

Littéral Un *littéral* est un prédicat appliqué à des termes, éventuellement précédé du symbole \neg . C'est donc une expression de la forme: $p(t_1, \dots, t_m)$ ou $\neg p(t_1, \dots, t_m)$, où p est un symbole de prédicat et les t_i sont des termes.

Un exemple de littéral: *plus – grand – que(age(pere(X)), age(mauricette))*

Atome Un littéral positif (c'est-à-dire non précédé du symbole \neg) est appelé un *atome*. Un littéral qui ne contient pas de variable est dit *clos* ou *complètement instancié*. Par exemple: *age(pere(jerome))* est un atome clos.

Formule Une *formule* est définie récursivement :

- Un littéral est une formule.
- Si α et β sont des formules alors $(\alpha \vee \beta)$, $(\alpha \wedge \beta)$, $(\alpha \rightarrow \beta)$, $(\alpha \leftrightarrow \beta)$ et $(\neg \alpha)$ sont des formules.
- Si v est une variable et α est une formule, alors $((\exists v)\alpha)$ et $((\forall v)\alpha)$ sont des formules.

Sous-formule Une sous-formule est une suite de symboles d'une formule, mais qui est elle-même une formule.

Variable libre et liée Une variable X est dite *liée* dans une formule ϕ s'il existe dans ϕ une sous-formule commençant par $((\forall X)$ ou par $((\exists X)$. Sinon, X est dite *libre*.

5.2.1.2 Le langage des clauses et des programmes logiques

Clause Une *clause* est une formule de type particulier: c'est une disjonction finie de littéraux dont toutes les variables sont quantifiées universellement, c'est-à-dire commandées par \forall . On simplifie en général l'écriture d'une clause en supprimant les quantificateurs \forall : toute variable doit donc être interprétée comme étant universellement quantifiée.

Clause de Horn Une clause de *Horn* est une clause qui a soit zéro soit un seul littéral positif.

Clause définie Une clause *définie* est une clause de Horn qui a exactement un littéral positif. Une clause définie s'écrit donc, quand on a supprimé les quantificateurs universels:

$$A \vee \neg B_1 \vee \dots \vee \neg B_m$$

On transforme cette notation en :

$$B_1 \wedge \dots \wedge B_m \rightarrow A$$

puis, en introduisant un nouveau connecteur et en changeant la notation de la conjonction :

$$A \leftarrow B_1, \dots, B_m$$

Le connecteur \leftarrow signifie l'implication du membre de gauche par le membre droit (donc l'inverse de \rightarrow). Cette notation est employée dans certaines versions du langage Prolog⁸. Dans la suite de ce chapitre, les clauses seront notées de cette manière.

8. La notation que nous employons est intermédiaire entre la notation logique et la notation la plus classique en Prolog, qui a été utilisée plus haut. Selon cette dernière, une clause s'écrit : $A :- B_1, \dots, B_m$.

Tête et corps de clause définie A , le seul littéral positif d'une clause définie est appelé *tête* de la clause et la conjonction B_1, \dots, B_m est le *corps* de la clause.

Une clause *unitaire* est une clause définie qui n'est composée que de son unique littéral positif A ; elle est donc notée $A \leftarrow$

Clause but Une *clause but* est une clause de Horn qui n'a aucun littéral positif; elle est donc notée: $\leftarrow B_1, \dots, B_m$

Programme logique défini Un programme logique défini, ou pour simplifier un *programme logique*, est un ensemble de clauses définies.

Par exemple:

```

enfant(X, Y) ← fille(X, Y)
enfant(X, Y) ← fils(X, Y)
fils(laurent, gaston) ←
fils(julien, laurent) ←
fille(laure, gaston) ←
fils(jerome, laure) ←
grand-parent(gaston, julien) ←
grand-parent(gaston, jerome) ←
grand-parent(gaston, julien) ←
grand-parent(gaston, jerome) ←
grand-parent(X, Y) ← enfant(Z, X), enfant(Y, Z)

```

Programme Prolog Un programme Prolog est un programme logique défini. Une requête est une clause but. Par exemple:

```

← enfant(Z, T)           % une clause but
enfant(X, Y) ← fille(X, Y)    % une clause
enfant(X, Y) ← fils(X, Y)
fils(laurent, gaston) ←      % un fait
fils(julien, laurent) ←
fille(laure, gaston) ←
fils(jerome, laure) ←

```

5.2.2 Système de preuve pour les langages de clauses

Étant donné un programme logique P , le but est de faire des *raisonnements* à partir de ce programme afin de savoir, par exemple, quels faits sont *VRAI* étant donné P .

En pratique, la preuve dans un programme Prolog se fait en utilisant la règle d'inférence logique dite de *modus ponens*, qui s'énonce informellement: « Si (α) et $(\alpha \rightarrow \beta)$ sont *VRAI*, alors (β) est *VRAI* », et se note classiquement:

$$\frac{\alpha \quad \wedge \quad (\alpha \rightarrow \beta)}{\beta}$$

L'algorithme de *résolution*, formalisé par Robinson [Rob65], est employé pour cette démonstration. Nous allons le présenter après avoir introduit les notions de substitution et d'unification.

5.2.2.1 La substitution

Une *substitution* est une liste finie de paires X_i/t_i , où X_i est une variable et t_i un terme. Si σ est la substitution $\{X_1/t_1, \dots, X_i/t_i, \dots, X_n/t_n\}$, l'ensemble des variables $\{X_1, \dots, X_n\}$ est noté $dom(\sigma)$.

Une substitution σ s'applique à une formule F en remplaçant chaque occurrence des variables de $dom(\sigma)$ par le terme correspondant, le résultat étant noté $F\sigma$.

5.2.2.2 L'unification

Si (s_i, t_i) est une paire de termes, un *unificateur* est une substitution σ telle que pour tout $i: s_i\sigma = t_i\sigma$.

Si un tel unificateur existe, on dit que les termes (s_i, t_i) peuvent s'unifier.

L'unificateur de deux littéraux $p(s_1, \dots, s_m)$ et $p(t_1, \dots, t_m)$ est un unificateur de l'ensemble des paires de termes $\{(s_i, t_i)\}$.

5.2.2.3 L'unificateur le plus général

L'*unificateur le plus général (upg)* est une substitution σ telle que pour tout unificateur θ , il existe une substitution γ telle que $\sigma\gamma = \theta$.

Il est démontré que l'*upg* de deux clauses est unique à un renommage de variables près.

5.2.2.4 La résolution

La résolution d'un programme logique consiste en une suite d'étapes, chacune construisant une nouvelle clause à partir de deux. Dans le cas d'un programme Prolog, dont les clauses sont d'une forme particulière (clauses définies et requêtes ne contenant que des littéraux), on peut appliquer une méthode adaptée : la *SLD-résolution*.

Une étape de SLD-résolution⁹ est définie comme suit. Soit deux clauses Prolog :

$$\begin{array}{ll} C_1: & H_1 \leftarrow A, a \\ C_2: & H_2 \leftarrow b \end{array}$$

où H_1 , a et b sont des conjonctions (éventuellement vides) d'atomes. A est un littéral quelconque du corps de C_1 , et a est le reste du corps de C_1 .

On dit que C_1 peut être résolue avec C_2 si H_2 et A s'unifient avec un *upg* θ (soit, $A\theta = H_2\theta$). Le résultat de l'étape de résolution est la clause suivante, appelée *résolvante* :

$$Res(C_1, C_2): \quad H_1\theta \leftarrow b\theta, a\theta$$

Par cette définition, on impose donc que le littéral résolu s'unifie avec la tête de C_2 et avec un littéral du corps de C_1 .

Sur la figure 5.1, on a $C_1 = fille(X, Y) \leftarrow parent(Y, X)$ et $C_2 = parent(claire, marie)$.

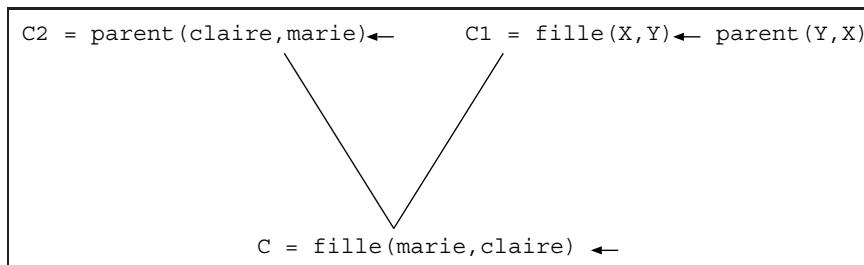


FIG. 5.1 – Une étape de la résolution.

9. SLD pour : *réolution linéaire avec fonction de sélection pour clauses définies*.

La clause C_1 peut se récrire sous la forme $C_1 = \text{fille}(X, Y) \vee \neg\text{parent}(Y, X)$. On peut alors appliquer la règle de résolution en choisissant :

- $L_1 = \neg\text{parent}(Y, X)$
- $L_2 = \text{parent}(\text{claire}, \text{marie})$
- $\theta = \{Y/\text{claire}, X/\text{marie}\}$

La clause résolvante C est alors l'union de :

- $(C_1 - \{L_1\})\theta = \text{fille}(\text{marie}, \text{claire})$
- $(C_2 - \{L_2\})\theta = \emptyset$

5.2.2.5 Résolution et interprétation d'un programme Prolog

Prolog est un langage conçu pour fournir une partie utile et efficace de la technique de démonstration par réfutation. Précisément, Prolog est un système basé sur un démonstrateur de théorème utilisant une forme particulière de résolution : la résolution linéaire avec fonction de sélection pour clauses définies (résolution SLD). Cette stratégie restreint le choix des clauses à chaque étape de résolution, ainsi que le choix du littéral qui est utilisé. Cela correspond à une recherche en profondeur cherchant d'abord à satisfaire chaque sous-but avant de passer au suivant. Si cette stratégie de recherche est systématique et (relativement) efficace, elle ne garantit malheureusement pas la terminaison des démonstrations. Prolog est en ce sens un démonstrateur sans garantie de complétude : des théorèmes vrais peuvent ne pas être démontrés parce que le démonstrateur tombe dans une branche infinie.

Dans le formalisme de Prolog, la connaissance, ou théorie, est transcrit sous forme de faits et de règles considérés comme des axiomes. Les requêtes sont exprimées comme des théorèmes (clauses de Horn négatives) dont on demande au démonstrateur de prouver leur validité dans la théorie.

5.3 La structuration de l'espace des hypothèses en logique des prédictats

Muni de la relation de couverture et d'une des relations de subsomption définies dans la section 5.1.2, il est possible d'envisager l'induction de programmes logiques comme une exploration de l'espace des hypothèses, guidée par les relations de subsomption. C'est ainsi par exemple que pour généraliser deux programmes, on cherchera leur plus petite généralisation suivant la relation de subsomption considérée, en faisant l'hypothèse qu'en généralisant minimalement on limite les risques de surgénéralisation. Le raisonnement est le même pour la recherche de spécialisations. Nous allons maintenant examiner les moyens effectifs de calculer des plus petites généralisations et des spécialisations maximales en fonction des relations de subsomption utilisées.

5.3.1 Le calcul de la *lgg* pour la θ -subsomption

Définition 5.1 (θ -subsomption)

On dit qu'une clause c_1 θ -subsume une clause c_2 si et seulement si il existe une substitution θ telle que $c_1\theta \subseteq c_2$. c_1 est alors une généralisation de c_2 par θ -subsomption.

Par exemple, la clause :

$$\text{pere}(X, Y) \leftarrow \text{parent}(X, Y), \text{etalon}(X)$$

θ -subsume la clause :

$$\text{pere}(\text{neral}, \text{aziza}) \leftarrow \text{parent}(\text{neral}, \text{aziza}), \text{etalon}(\text{neral}), \text{jument}(\text{aziza})$$

avec $\theta = \{(X = \text{neral}), (Y = \text{aziza})\}$

Plotkin a introduit la notion de *moindre généralisé* ou encore de *plus petit généralisé* [Plo70], un opérateur de généralisation qui permet de généraliser deux clauses. Le calcul de la généralisation la moins générale de deux clauses est défini par les règles suivantes :

- La généralisation la moins générale de deux termes t_1 et t_2 , notée $\text{lgg}(t_1, t_2)$ (*least general generalization*), est une variable si :
 - au moins un des deux termes est une variable,
 - un des deux termes est une constante, et $t_1 \neq t_2$,
 - t_1 et t_2 sont deux termes fonctionnels construits sur des symboles de fonction différents.
- Si $t_1 = f(c_1, \dots, c_n)$ et $t_2 = f(d_1, \dots, d_n)$, alors

$$\text{lgg}(t_1, t_2) = f(\text{lgg}(c_1, d_1), \dots, \text{lgg}(c_n, d_n))$$

Pour appliquer cette règle, il faut prendre soin de vérifier que si θ_1 et θ_2 sont les deux substitutions telles que $\text{lgg}(t_1, t_2)\theta_i = t_i$ ($i \in \{1, 2\}$), alors il ne doit pas exister deux variables distinctes X et Y telles que $X\theta_1 = Y\theta_1$ et $X\theta_2 = Y\theta_2$.

Par exemple, la généralisation la moins générale de $f(a, b, a)$ et de $f(b, a, b)$ est $f(X, Y, X)$ et non $f(X, Y, Z)$.

- La lgg de deux littéraux construits sur le même symbole de prédicat est donnée par :

$$\text{lgg}(p(t_1, \dots, t_n), p(s_1, \dots, s_n)) = p(\text{lgg}(t_1, s_1), \dots, \text{lgg}(t_n, s_n))$$

- Enfin, la lgg de deux clauses $l_0 \leftarrow l_1, \dots, l_n$ et $m_0 \leftarrow m_1, \dots, m_n$ est une clause qui a pour tête $\text{lgg}(l_0, m_0)$, et pour corps l'ensemble des $\text{lgg}(l_i, m_i)$ pour tout couple (l_i, m_i) de littéraux de même signe et de même prédicat.

Pour illustrer la lgg , considérons les deux scènes constituées d'objets géométriques, représentées sur la figure 5.2 [MB96].

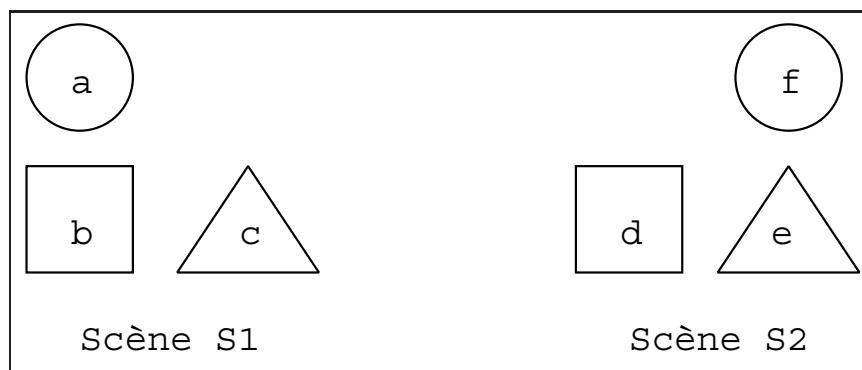


FIG. 5.2 – *La généralisation la moins générale.*

La première scène représente trois objets a , b , et c . L'objet a est un cercle, b est un carré et c est un triangle. Les objets sont placés de telle façon que a est au-dessus de b , et que b est situé

à gauche de c . La deuxième scène représente trois objets d , e , et f décrits de manière similaire. Les deux scènes S_1 et S_2 sont décrites par les deux clauses suivantes qui décrivent le placement de trois figures géométriques :

$$\begin{array}{ll} \textit{scene}(s1) \leftarrow & \textit{scene}(s2) \leftarrow \\ \textit{sur}(s1, a, b), & \textit{sur}(s2, f, e), \\ \textit{a_gauche}(s1, b, c), & \textit{a_gauche}(s2, d, e), \\ \textit{cercle}(a), & \textit{cercle}(f), \\ \textit{carre}(b), & \textit{carre}(d), \\ \textit{triangle}(c). & \textit{triangle}(e). \end{array}$$

La lgg des clauses représentant les deux scènes de la figure 5.2 est la clause suivante :

$$\begin{aligned} \textit{scene}(S) \leftarrow \\ \textit{sur}(S, A, B), \textit{a_gauche}(S, C, D), \\ \textit{cercle}(A), \textit{carre}(C), \textit{triangle}(D). \end{aligned}$$

Cette clause généralise les deux clauses précédentes et traduit le fait que dans les deux scènes, l'objet circulaire se trouve au-dessus d'un autre objet, et que l'objet carré est à gauche de l'objet triangulaire.

5.3.2 Le calcul de $rlgg$ pour la θ -subsomption relative

On définit la plus petite généralisation relative ($rlgg$), (*relative least general generalization*) de deux clauses en utilisant la θ -subsomption relative de manière tout à fait analogue.

Définition 5.2 (θ -subsomption relative)

Soit P un programme logique. Une clause c_1 θ -subsume une clause c_2 relativement à P si et seulement si il existe une substitution θ telle que $P \models c_1\theta \rightarrow c_2$.

Définition 5.3 ($rlgg$ de deux exemples)

Soient deux exemples e_1 et e_2 qui sont des atomes liés. Soit une théorie du domaine $T = a_1 \wedge \dots \wedge a_n$ où les a_i sont également des atomes liés. La $rlgg$ de e_1 et e_2 est donnée par :

$$rlgg(e_1, e_2) = lgg(e_1 \leftarrow T, e_2 \leftarrow T)$$

avec :

$$\begin{aligned} e_i \leftarrow T &= \neg T \vee e_i \\ &= \neg(a_1 \wedge \dots \wedge a_n) \vee e_i \\ &= \neg a_1 \vee \dots \vee \neg a_n \vee e_i \end{aligned}$$

Nous illustrons le calcul de la $rlgg$ de deux clauses en considérant l'exemple du tri rapide (*quick sort*) [MF90]. La théorie du domaine, qui traduit la connaissance initiale sur le problème

est constituée d'instances de littéraux construits sur les prédictats *partition/4* et *append/3* :

```

partition(1, [], [], []) ←
partition(2, [4, 3, 1, 0], [1, 0], [4, 3]) ←
...
append([], [1], [1]) ←
append([0, 1], [2, 3, 4], [0, 1, 2, 3, 4]) ←
...

```

Supposons que l'on dispose d'un certain nombre d'exemples de tris réalisés à l'aide du prédictat *qsort/2* :

```

qsort([], []) ←
qsort([1, 0], [0, 1]) ←
qsort([4, 3], [3, 4]) ←
...

```

On dispose aussi de deux exemples positifs e_1 et e_2 dont on désire calculer la *rlgg* d'après la définition 5.3.

$$\begin{aligned} e_1 &= qsort([1], [1]) \\ e_2 &= qsort([2, 4, 3, 1, 0], [0, 1, 2, 3, 4]) \end{aligned}$$

La *rlgg* de e_1 et e_2 est donnée par :

$$rlgg(e_1, e_2) = lgg(e_1 \leftarrow T, e_2 \leftarrow T)$$

où T est la conjonction de tous les atomes liés de la théorie du domaine et des exemples. On a donc :

$$\begin{aligned} e_1 \leftarrow T &= qsort([1], [1]) \leftarrow \\ &\quad append([], [1], [1]), append([0, 1], [2, 3, 4], [0, 1, 2, 3, 4]), \dots, \\ &\quad partition(1, [], [], []), partition(2, [4, 3, 1, 0], [1, 0], [4, 3]), \dots, \\ &\quad qsort([], []), qsort([1, 0], [0, 1]), qsort([4, 3], [3, 4]) \\ &\quad \dots \end{aligned}$$

et

$$\begin{aligned} e_2 \leftarrow T &= qsort([2, 4, 3, 1, 0], [0, 1, 2, 3, 4]) \leftarrow \\ &\quad append([], [1], [1]), append([0, 1], [2, 3, 4], [0, 1, 2, 3, 4]), \dots, \\ &\quad partition(1, [], [], []), partition(2, [4, 3, 1, 0], [1, 0], [4, 3]), \dots, \\ &\quad qsort([], []) \\ &\quad qsort([1, 0], [0, 1]), qsort([4, 3], [3, 4]) \\ &\quad \dots \end{aligned}$$

La *rlgg* des atomes e_1 et e_2 est donc définie par :

```

 $rlgg(e_1, e_2) = qsort([A|B], [C|D]) \leftarrow$ 
   $append(E, [A|F], [C|D]), append([], [1], [1]),$ 
   $append(G, [H|J], [J|K]), append([0, 1], [2, 3, 4], [0, 1, 2, 3, 4]),$ 
   $partition(A, B, L, M), partition(1, [], [], []),$ 
   $partition(H, N, O, P), partition(2, [4, 3, 1, 0], [1, 0], [4, 3]),$ 
   $qsort(L, E), qsort([], []),$ 
   $qsort(O, G), qsort([1, 0], [0, 1]),$ 
   $qsort(M, F), qsort(P, I), qsort([4, 3], [3, 4]),$ 
  ...

```

La construction de la *rlgg* pose un certain nombre de problèmes. En effet, comme nous l'avons déjà évoqué, les atomes de la théorie du domaine T utilisés dans le calcul de la *rlgg* doivent être des atomes liés. Cela impose une définition en extension de la théorie du domaine, ce qui n'est pas concevable pour la majorité des problèmes à résoudre. Pour résoudre ce problème, [Bun88] suggère de calculer cette définition en extension à partir du plus petit modèle de Herbrand de la connaissance initiale exprimée en intension. Cette méthode n'est pas entièrement satisfaisante, puisqu'elle risque d'omettre un certain nombre d'instances de la théorie du domaine et fausser le résultat de l'algorithme d'apprentissage. Plotkin, quant à lui, propose une méthode pour supprimer les littéraux logiquement redondants [Plo71a] [Plo71b]. Malheureusement, la détection des littéraux redondants est coûteuse puisqu'elle nécessite la mise en place de techniques de preuve de théorème. De plus, les clauses débarassées de leurs littéraux redondants peuvent encore contenir un grand nombre de littéraux.

5.3.3 Le calcul de *lgg* pour la résolution inverse

Plutôt que de définir la subsomption par référence à la sémantique, Muggleton a proposé de la définir à partir de la technique de preuve, qui en logique des prédictats, est le principe de résolution de Robinson utilisé dans Prolog. Dans ce cadre, on dira qu'une clause subsume une autre clause si elle permet la déduction de celle-ci par SLD-résolution. Plus formellement :

Définition 5.1 (SLD-subsomption)

Soit \mathcal{P} un programme logique; une clause C est plus générale qu'une clause D au sens de la SLD-subsomption relativement à \mathcal{P} ssi $C, \mathcal{P} \vdash_{SLD} D$.

En dehors du fait qu'elle prend en compte la théorie du domaine sous la forme du programme \mathcal{P} , l'un des intérêts de cette définition est qu'elle induit, comme la θ -subsomption, des opérations syntaxiques sur les programmes qui sont les fondements de la technique d'apprentissage par inversion de la résolution. Par ailleurs, la justesse de la SLD-résolution garantit que ce qui dérive d'une théorie par SLD-résolution est une conséquence logique de cette théorie. En bref, si $C, \mathcal{P} \vdash_{SLD} D$, alors $C, \mathcal{P} \models D$, ce qui fait de la SLD-subsomption un cas particulier « constructif » de l'implication relative à une théorie.

On peut alors dériver analytiquement l'inversion de la résolution à partir de la règle de résolution exprimée par l'équation 5.1. Tout d'abord, θ peut toujours s'écrire sous la forme d'une composition de substitutions θ_1 et θ_2 , avec θ_i contenant les variables de C_i . L'équation s'écrit donc :

$$C = (C_1 - \{L_1\})\theta_1 \cup (C_2 - \{L_2\})\theta_2 \quad (5.1)$$

On restreint l'inversion de résolution à l'inférence de clauses C_2 qui ne contiennent aucun littéral en commun avec C_1 .

$$C - (C_1 - \{L_1\})\theta_1 = (C_2 - \{L_2\})\theta_2$$

or $L_1\theta_1 = \neg L_2\theta_2$ donc $L_2 = \neg L_1\theta_1\theta_2^{-1}$, on obtient ainsi :

$$C_2 = (C - (C_1 - \{L_1\})\theta_1)\theta_2^{-1} \cup \{\neg L_1\theta_1\theta_2^{-1}\} \quad (5.2)$$

On note le non-déterminisme de cet opérateur, notamment concernant le choix de la clause C_1 , et des substitutions θ_1 et θ_2 .

Dans le cadre de la PLI, S. Muggleton et W. Buntine ont donc eu l'idée d'inverser la résolution classique utilisée en programmation logique. Dans [Mug87], S. Muggleton introduit quatre règles de résolution inverse. La notation $\frac{A}{B}$ s'interprète comme : on peut déduire A de B :

Absorbtion (opérateur V)

$$\frac{q \leftarrow A \quad p \leftarrow A, B}{q \leftarrow A \quad p \leftarrow q, B}$$

Identification (opérateur V)

$$\frac{p \leftarrow A, B \quad p \leftarrow A, q}{q \leftarrow B \quad p \leftarrow A, q}$$

Intra construction (opérateur W)

$$\frac{p \leftarrow A, B \quad p \leftarrow A, C}{q \leftarrow B \quad p \leftarrow A, q \quad q \leftarrow C}$$

Inter construction (opérateur W)

$$\frac{p \leftarrow A, B \quad q \leftarrow A, C}{p \leftarrow r, B \quad r \leftarrow A \quad q \leftarrow r, C}$$

Dans ces règles, les lettres minuscules représentent des atomes et les lettres majuscules des conjonctions d'atomes. Les règles d'absorbtion et d'identification inversent une étape de résolution. La figure 5.3 montre comment l'opérateur V inverse une étape de la résolution.

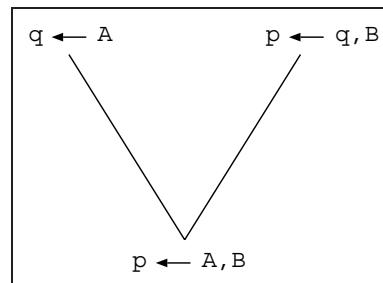


FIG. 5.3 – L'opérateur V d'absorbtion.

La figure 5.4 illustre par un exemple le comportement de l'opérateur V en montrant plusieurs étapes de résolution inversée. La théorie du domaine est ici constituée des clauses b_1 et b_2 et on dispose d'une observation e_1 . L'opérateur V permet, à partir de l'exemple e_1 et de la clause b_2 , d'induire la clause c_1 . La clause c_1 permet ensuite, avec la clause b_1 , d'induire la clause c_2 .

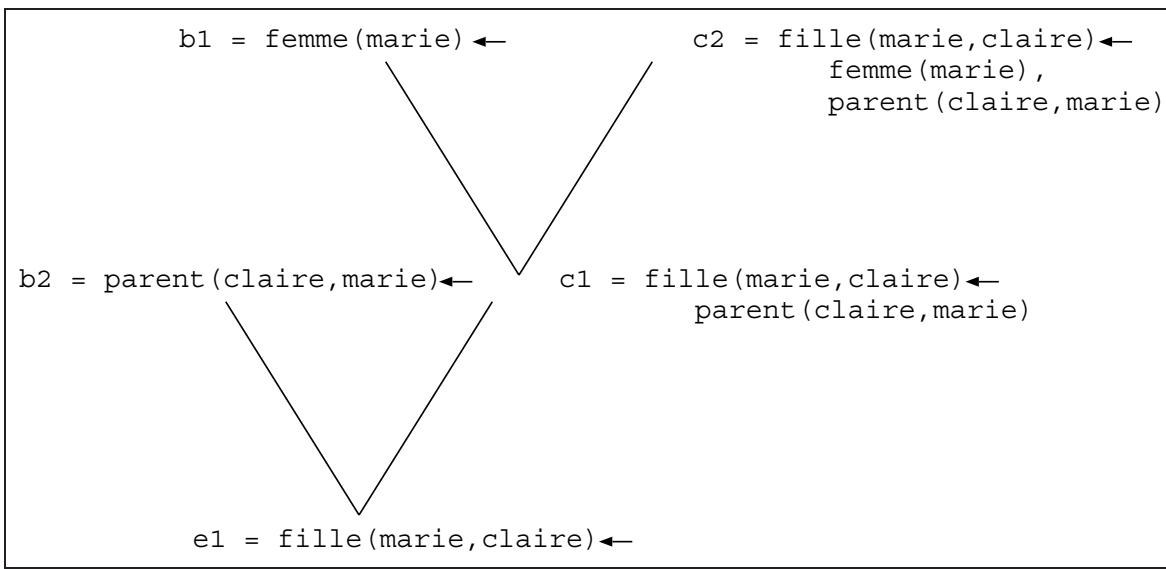


FIG. 5.4 – La résolution inverse.

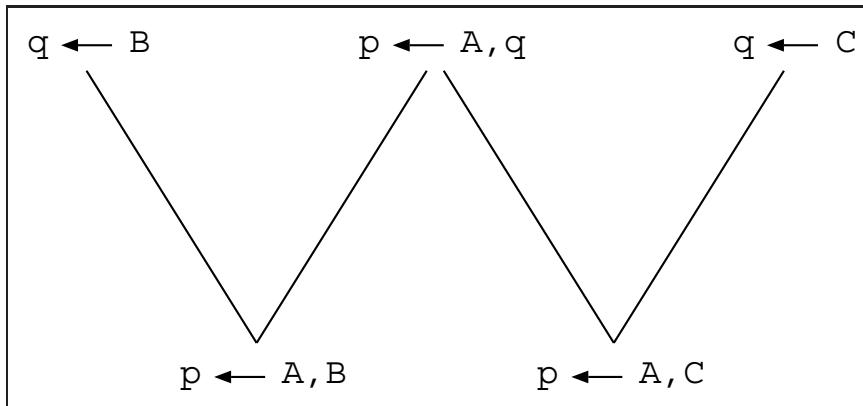


FIG. 5.5 – L'opérateur W d'intra-construction.

Les opérateurs d'intra construction et d'inter construction, appelés aussi opérateurs W (figure 5.5) résultent de la combinaison de deux opérateurs V qui représentent chacun une étape inverse de résolution.

Ces règles d'inférence présentent la particularité d'introduire un nouveau prédicat qui n'apparaissait pas dans les préconditions des règles. Par exemple, l'opérateur W de la figure 5.5 introduit le nouveau prédicat q . On parle alors d'*invention de prédicat*, utilisée par exemple dans le système CIGOL [MB88b]. L'invention de prédicat est bien illustrée par l'exemple de la figure 5.6.

En effet, dans cet exemple, on dispose des deux clauses :

$$\begin{aligned} \min(X, [s(X) \parallel Z]) &\leftarrow \min(X, Z) \\ \min(X, [s(s(X)) \parallel Z]) &\leftarrow \min(X, Z) \end{aligned}$$

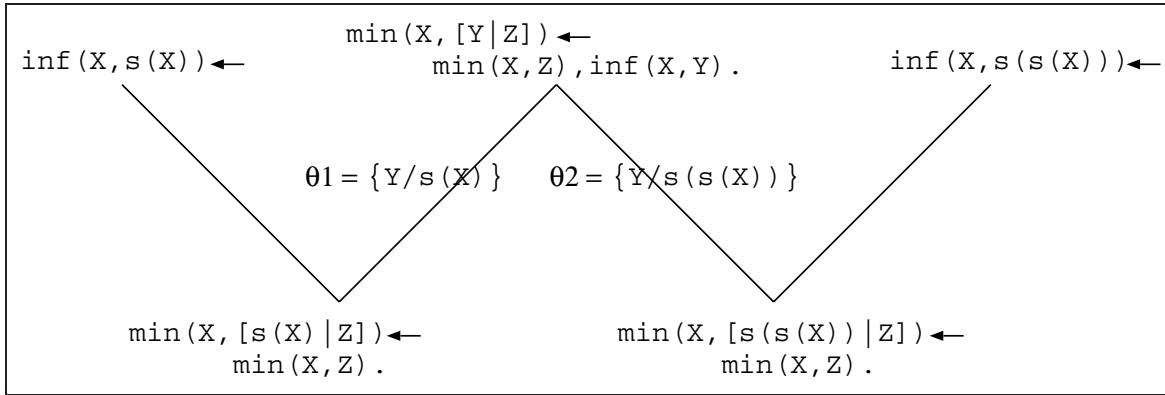


FIG. 5.6 – L'intra construction.

et l'application de l'opérateur W permet d'induire les trois clauses :

$$\begin{aligned} \text{inf}(X, s(X)) &\leftarrow \\ \text{min}(X, [Y||Z]) &\leftarrow \text{min}(X, Z), \text{inf}(X, Y) \\ \text{inf}(X, s(s(X))) &\leftarrow \end{aligned}$$

dans lesquelles on constate que le prédicat $\text{inf}/2$ a été inventé.

L'inversion de la résolution est une opération non déterministe. Typiquement, à chaque étape de l'inversion de la résolution, plusieurs généralisations d'une clause peuvent être réalisées, en fonction du choix de la clause avec laquelle elle est résolue, et de la substitution inverse employée. Pour surmonter ce problème de non-déterminisme, Muggleton a proposé, dans un cadre unifiant la plus petite généralisation relative ($rlgg$) et la résolution inverse, que ce soit l'inversion de résolution la plus spécifique qui soit choisie dans le processus de généralisation, en utilisant la substitution inverse la plus spécifique à chaque pas de l'inversion de résolution.

5.4 L'exploration de l'espace des hypothèses

La notion de couverture d'un exemple par une hypothèse, la relation de subsomption entre hypothèses et les opérateurs permettant de construire les plus petites généralisations (ou spécialisations) autorisent à envisager une approche de l'apprentissage par la détermination de l'espace des versions, comme cela a été montré dans la chapitre 4. Cependant, en raison de la taille de l'espace des versions en programmation logique inductive, les programmes d'induction ne cherchent pas à construire l'espace des versions, même à l'aide des bornes que sont le S-set et le G-set. Ils cherchent « seulement » à trouver une solution dans l'espace des versions. Pour ce faire, ils parcourront l'espace des hypothèses en suivant les directions de généralisation et de spécialisation, en essayant d'élaguer au maximum l'espace des versions et en se guidant par des biais heuristiques tentant d'accélérer la recherche.

Les programmes de PLI s'imposent en général de respecter certaines contraintes connues sous le nom de *sémantique normale* [MD94]. Elles correspondent à une reformulation des conditions de cohérence (complétude et correction) d'une hypothèse avec les exemples positifs et négatifs.

Étant donnés une *théorie du domaine T*, c'est-à-dire un ensemble de clauses qui reflètent la connaissance *a priori* sur domaine du problème, un ensemble d'exemples E , (avec l'ensemble $E = E^+ \cup E^-$ constitué d'exemples positifs et négatifs), la programmation logique inductive

cherche à induire une hypothèse H , composée d'un ensemble de clauses, telle que les quatre conditions suivantes soient respectées :

Définition 5.2 (Sémantique normale)

- *satisfiabilité a priori* : $T \wedge E^- \not\models \square$
- *satisfiabilité a posteriori* : $T \wedge H \wedge E^- \not\models \square$
- *nécessité a priori* : $T \not\models E^+$
- *condition a posteriori (complétude)* : $T \wedge H \models E^+$

La condition de satisfiabilité *a priori* permet de s'assurer que les exemples négatifs ne peuvent pas être déduits de la connaissance initiale.

La condition de consistance permet de vérifier qu'à partir de l'hypothèse induite et de la théorie du domaine, on ne peut pas prouver d'exemple négatif.

Il est également nécessaire que les exemples positifs ne puissent pas être déduits de la connaissance T (*nécessité a priori*).

L'hypothèse induite doit quant à elle permettre, avec la connaissance initiale, de prouver les exemples positifs (*condition a posteriori*).

En fait, dans la plupart des systèmes de PLI, la théorie (ou connaissance initiale) T du domaine et l'hypothèse H à induire sont exprimées sous forme de clauses définies. Dans ce cas, on parle de sémantique définie¹⁰. Seules les quatre conditions suivantes doivent alors être remplies :

Définition 5.3 (Sémantique définie)

- $\forall e \in E^-, e \text{ est faux dans } \mathcal{M}(T)$
- $\forall e \in E^-, e \text{ est faux dans } \mathcal{M}(T \wedge H)$
- $\exists e \in E^+, \text{ tel que } e \text{ est faux dans } \mathcal{M}(T)$
- $\forall e \in E^+, e \text{ est vrai dans } \mathcal{M}(T \wedge H)$

Pour simplifier et pour couvrir les cas pratiques, la majorité des systèmes de programmation logique inductive est basée sur le cas particulier de la sémantique définie, dans lequel tous les exemples sont des faits liés.

5.4.1 Le squelette des algorithmes de PLI

L'algorithme générique de la PLI utilise deux fonctions : la fonction **effacer** influence la stratégie de recherche qui peut alors être effectuée soit en largeur d'abord, soit en profondeur d'abord, ou selon une autre stratégie ; la fonction **choisir** détermine les règles d'inférence à appliquer à l'hypothèse H .

Définition 5.4 (Règle d'inférence déductive r)

Elle fait correspondre une conjonction de clauses S à une conjonction de clauses G telle que $G \models S$. r est une règle de spécialisation.

10. Si une théorie T est constituée de clauses définies, elle possède une fermeture syntaxique appelée le plus petit modèle de Herbrand dans lequel toute formule logique est soit vraie, soit fausse.

Algorithme 5.1 Algorithme générique de PLI

```

Initialiser  $H$ 
tant que la condition d'arrêt de  $H$  n'est pas remplie faire
    effacer un élément  $h$  de  $H$ 
    choisir des règles d'inférence  $r_1, \dots, r_k$  à appliquer à  $h$ 
    Appliquer les règles  $r_1, \dots, r_k$  à  $h$  pour obtenir  $h_1, \dots, h_n$ 
    Ajouter  $h_1, \dots, h_n$  à  $H$ 
    élaguer  $H$ 
fin tant que

```

Définition 5.5 (Règle d'inférence inductive r)

Elle fait correspondre une conjonction de clauses G à une conjonction de clauses S telle que $G \models S$. r est une règle de généralisation.

5.4.1.1 Stratégies de recherche

La stratégie de recherche est un choix fondamental pour le concepteur d'un système de PLI :

- Les systèmes CIGOL [MB88b], CLINT [De 92], et GOLEM [MF90] sont représentatifs de la classe des *systèmes ascendants*. Ils considèrent les exemples et la théorie du domaine et généralisent itérativement l'hypothèse recherchée en appliquant des règles d'inférence inductives.
- Les systèmes FOIL [Qui90], [QC95] MOBAL [KW92], PROGOL [Mug95], et CLAUDIEN [DB93], [DVD96], [DD97] sont quant à eux représentatifs de la classe des *systèmes descendants*. Ils considèrent d'abord l'hypothèse la plus générale qu'ils cherchent à spécialiser itérativement en appliquant des règles d'inférence déductives.

5.4.1.2 L'élagage de l'espace de recherche

La fonction **élaguer** détermine quelles hypothèses doivent être enlevées de l'ensemble H . La généralisation et la spécialisation sont à la base de deux cas d'élagage de l'ensemble des hypothèses. Cet élagage est le même que celui pratiqué dans la construction de l'espace des versions par l'algorithme d'élimination des candidats.

- Si $B \wedge H$ n'implique pas logiquement un exemple positif e^+ , c'est-à-dire $B \wedge H \not\models e^+$, alors aucune spécialisation de H ne pourra impliquer e^+ , et toutes les spécialisations de H peuvent donc être élaguées de l'espace de recherche.
- Si un exemple négatif e^- est couvert, c'est-à-dire $B \wedge H \wedge e^- \models \square$, alors toutes les généralisations de H peuvent être élaguées puisqu'elles sont également inconsistantes avec $B \wedge E$.

L'exemple de la figure 5.7 illustre le premier cas.

Supposons que la base de connaissances contienne les faits :

```

pere(gaston,francois) ←
pere(gaston,laurent) ←
pere(laurent,julien) ←

```

et que l'ensemble des exemples positifs contienne l'exemple :

```

oncle(francois,julien) ←

```

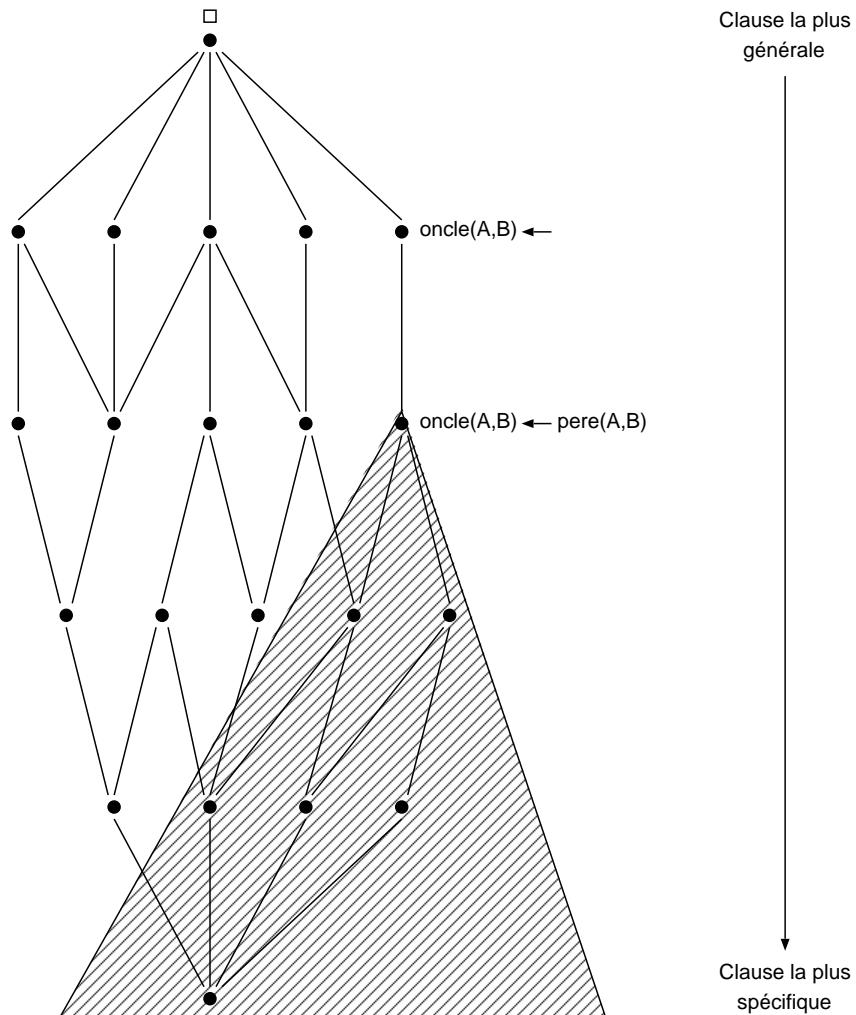


FIG. 5.7 – L'élagage de l'espace de recherche par la relation de subsomption.

on peut élaguer toutes les clauses plus spécifiques que la clause $\text{oncle}(A, B) \leftarrow \text{pere}(A, B)$ (zone hachurée de la figure 5.7).

Mais l'espace de recherche n'est pas élagué assez efficacement par cette propriété, qui rappelons-le, découle directement de la relation de subsomption entre hypothèses. Il faut donc définir des biais supplémentaires limitant davantage l'espace de recherche.

5.4.2 Les biais de recherche dans l'espace d'hypothèses

L'espace de recherche peut être très vaste, voire infini. L'implantation effective d'un système de PLI nécessite donc l'utilisation de biais déclaratifs. Les biais imposent des contraintes ou des restrictions sur les hypothèses que l'algorithme peut considérer et définissent donc l'espace de recherche. Le rôle du biais est double :

- réduire la taille de l'espace de recherche pour diminuer le temps de réponse du système ;
- garantir une certaine qualité de l'apprentissage en interdisant au système de considérer certaines hypothèses inutiles.

On distingue deux classes de biais déclaratifs : les biais syntaxiques, aussi appelés biais de langage, et les biais sémantiques.

5.4.2.1 Les biais syntaxiques

Un biais syntaxique permet de définir l'ensemble des hypothèses envisageables en spécifiant explicitement leur syntaxe. Le nombre des hypothèses à considérer peut en particulier être réduit par les méthodes suivantes :

- limiter le nombre de littéraux des clauses induites ;
- limiter le nombre de variables apparaissant dans une clause ;
- limiter la profondeur de la récursion dans les termes ;
- n'autoriser que les clauses *range-restricted*, c'est-à-dire dont l'ensemble des variables de la tête est inclus dans l'ensemble des variables du corps.

5.4.2.2 Les biais sémantiques

Si les biais syntaxiques restreignent l'espace de recherche en imposant la syntaxe des hypothèses, les biais sémantiques imposent des restrictions sur le sens des hypothèses. La définition de *types* et de *modes* est utilisée depuis de nombreuses années en programmation logique afin de permettre une meilleure analyse statique des programmes et ainsi d'augmenter leur efficacité. Depuis leur introduction dans le système MIS [Sha83], il est devenu courant d'utiliser également les modes et les types en PLI. Ils constituent alors une façon de définir des biais sémantiques. En effet, si un littéral contient une variable L de type t , toutes les autres occurrences de cette variable L dans la même clause devront également être de type t . Ceci permet d'élaguer de l'espace de recherche des clauses incorrectement typées. De même, la déclaration de modes permet de restreindre la taille de l'espace de recherche. Par exemple, l'utilisation du prédicat *concat/3* associée à la déclaration de mode suivante :

$$\text{concat}(+L1, +L2, -L3)$$

où ‘+’ signifie argument en entrée (devant être instanciés), et ‘-’ argument en sortie, permet de rejeter de l'espace de recherche toutes les clauses utilisant le prédicat *concat* sur des variables ne respectant pas ce mode, comme par exemple :

$$\text{concat}(L1, L2, [a, b, c])$$

Voici quelques façons d'utiliser les modes sur les variables des prédicats :

- Toutes les variables en entrée dans la tête de la clause doivent se retrouver dans le corps de clause.
- Les variables en sortie dans la tête doivent être présentes dans le corps.
- On peut aussi forcer toutes les variables en sortie dans le corps soit à servir de résultat intermédiaire (et donc d'être en entrée dans un autre prédicat), soit à servir à définir la sortie du prédicat de la tête (et donc d'apparaître en sortie dans la tête).
- Il est également envisageable d'interdire qu'une même variable apparaisse plusieurs fois en sortie de différents prédicats dans le corps de la clause.

Le système PROGOL [Mug95],[Rob97] permet par exemple de déclarer des modes sur les variables des littéraux afin de restreindre le nombre potentiel de clauses dans l'espace de recherche. De plus, le système distingue les prédicats pouvant apparaître en tête de clause de ceux admis dans le corps. Les littéraux qui peuvent apparaître en tête sont déclarés avec la directive *modeh*.

Les littéraux autorisés dans les corps de clauses sont eux déclarés grâce à la directive *modeb*. Par exemple, la déclaration

$$\text{modeh}(1, \text{plus}(+\text{int}, +\text{int}, -\text{int}))$$

spécifie que la tête des clauses de l'espace de recherche sera constituée du prédicat *plus/3* avec trois arguments de type entier, les deux premiers arguments étant en mode entrée et le dernier en mode sortie.

5.5 Deux exemples de systèmes de PLI

5.5.1 Un système empirique descendant : FOIL

Le système FOIL¹¹ [Qui90] a été développé par J.R. Quinlan et a ensuite inspiré d'autres systèmes parmi lesquels FOCL [PK92], FOIDL [MC95a], MFoil [Dze93], ICN et MULT_ICN [MV95]. FOIL cherche à induire un programme logique qui couvre tous les exemples positifs du concept à apprendre, et aucun exemple négatif. Chaque clause apprise est construite par spécialisation successive de la clause la plus générale par ajout d'un nouveau littéral. Le littéral ajouté est choisi en calculant un *gain d'information* de façon similaire à la construction d'un arbre de décision (voir le chapitre 11). L'utilisation de FOIL impose une représentation en extension de la théorie du domaine. FOIL permet de définir des biais syntaxiques (limitation du nombre de variables apparaissant dans chaque clause, taux minimal de couverture des clauses, etc.). Il est également possible de préciser les modes des arguments des prédictats.

5.5.1.1 L'algorithme

Algorithme 5.2 Algorithme FOIL

```

 $P \leftarrow \emptyset$ 
 $Pos \leftarrow$  exemples positifs
tant que  $Pos$  est non vide faire
     $Neg \leftarrow$  exemples négatifs
     $C = q(X_1, \dots, X_n)$ 
    tant que  $Neg$  est non vide faire
        Ajouter le littéral de meilleur gain au corps de  $C$ 
        Retirer de  $Neg$  les exemples négatifs non couverts par  $C$ 
    fin tant que
    Ajouter la clause apprise  $C$  à  $P$ 
    Retirer de  $Pos$  les exemples couverts par  $C$ 
fin tant que
Retourner le programme appris  $P$ 

```

L'algorithme ci-dessus est l'algorithme de base du système FOIL. La boucle la plus externe permet de construire des clauses tant que tous les exemples ne sont pas couverts. La boucle interne construit une clause en ajoutant un à un des littéraux qui ont le gain le plus élevé. Pour la

11. Le système FOIL est accessible par ftp à l'adresse <ftp://ftp.cs.su.oz.au/pub/foil6.sh>

construction de chaque clause à l'étape i , FOIL gère deux ensembles de tuples T_i^+ et T_i^- . Chaque élément de ces ensembles est une instance liée de la clause en construction correspondant à un exemple couvert. Les tuples de T_i^+ correspondent à des exemples positifs, et ceux de T_i^- à des exemples négatifs. À chaque étape, les ensembles T_i^+ et T_i^- sont calculés à partir des ensembles T_{i-1}^+ et T_{i-1}^- de l'étape précédente. La fonction de gain utilisée pour le choix du littéral L à ajouter lors du passage à une nouvelle étape est calculée à partir du nombre d'éléments des différents ensembles de tuples :

$$gain(L) = n_i^+ \left(\log_2 \frac{\|T_i^+\|}{\|T_i^+\| + \|T_i^-\|} - \log_2 \frac{\|T_{i+1}^+\|}{\|T_{i+1}^+\| + \|T_{i+1}^-\|} \right)$$

où n_i^+ est le nombre d'exemples positifs couverts par la clause en construction, et les T_{i+1} sont les ensembles de tuples considérés lorsqu'on a ajouté le littéral L .

L'un des problèmes majeurs de FOIL intervient lors de la construction d'une clause. En effet, à chaque ajout d'un littéral dans la clause, il se peut que le nombre de littéraux candidats soit très grand. Or pour chacun d'entre eux, FOIL doit calculer les ensembles de tuples positifs et négatifs pour être en mesure d'évaluer le gain. Ceci peut être préjudiciable à l'efficacité du système.

Du fait de sa méthode d'apprentissage, FOIL permet de traiter des données bruitées. En effet, il peut considérer qu'une certaine partie des exemples est bruitée, et arrêter la construction de la clause lorsqu'un certain pourcentage d'exemples est couvert.

5.5.1.2 Une illustration du fonctionnement

Nous illustrons l'utilisation du système FOIL par l'apprentissage d'une définition du concept *oncle*. On dispose de la connaissance suivante sur les liens de parenté au sein d'une famille.

```
/* valeurs possibles pour les arguments des predicats */
P: alfred,michel,remi,franck,charles,paul.

oncle(P,P)
/* exemples positifs pour le predicat oncle */
remi,paul           remi,franck          michel,charles
;
/* exemples negatifs pour le predicat oncle */
alfred,alfred       alfred,michel      alfred,remi
alfred,franck       alfred,charles    alfred,paul
michel,alfred       michel,michel    michel,remi
michel,franck       michel,charles  remi,alfred
remi,michel         remi,charles    remi,remi
franck,paul         franck,alfred   franck,michel
franck,remi         franck,charles franck,charles
charles,paul        charles,alfred  charles,michel
charles,remi        charles,franck  charles,charles
paul,paul           paul,alfred    paul,michel
paul,remi          paul,franck   paul,charles
.

*pere(P,P)
/* exemples positifs pour le predicat pere */
michel,paul         alfred,michel   alfred,remi
michel,franck       remi,charles
```

```

;
/* exemples negatifs pour le predicat pere */
alfred,alfred      alfred,franck      alfred,charles
alfred,paul        michel,alfred      michel,michel
michel,remi        michel,charles    remi,alfred
remi,michel        remi,remi         remi,franck
remi,paul          franck,alfred     franck,michel
franck,remi        franck,franck     franck,charles
franck,paul        charles,alfred   charles,michel
charles,remi       charles,franck   charles,charles
charles,paul       paul,alfred      paul,michel
paul,remi          paul,franck      paul,charles
paul,paul

```

FOIL apprend le prédictat `oncle/2` défini par la clause:

```
oncle(A,B) :- pere(C,A), pere(D,B), pere(C,D), A<>D.
```

5.5.2 Un système empirique ascendant : PROGOL

PROGOL¹² est un système de PLI basé sur l’implication inverse [Mug95]. Il cherche à induire des clauses grâce à une approche descendante. PROGOL possède la particularité de calculer, avant de construire une clause, la clause la plus spécifique couvrant l’exemple considéré. Cette clause la plus spécifique constitue une borne lors du processus de construction de la clause à induire. La clause finale est calculée grâce à un algorithme A^* intégrant une mesure de compression que PROGOL cherche à maximiser.

Des déclarations de modes et de types permettent de définir des biais sémantiques. Il est nécessaire de spécifier le littéral à employer en tête de clause (en utilisant la directive `modeh`), et les littéraux qu’il est possible d’utiliser dans le corps des clauses (en utilisant la directive `modeb`).

PROGOL permet de définir la théorie du domaine en intension sous la forme de clauses Prolog. Notons que ceci est un apport significatif par rapport à FOIL avec lequel il était nécessaire de définir la connaissance initiale en extension. Les exemples positifs sont exprimés par des clauses définies, et les exemples négatifs sont des négations de faits instanciés. [Rob97] expose les différents paramètres du système.

5.5.2.1 L’algorithme

Le principe du fonctionnement de PROGOL est donné par l’algorithme 5.3.

5.5.2.2 Une illustration du fonctionnement

Considérons par exemple, l’apprentissage du tri rapide par PROGOL. Etant donné le programme suivant :

```

% Declaration des modes pour le litteral en tete de clause
:- modeh(1,qsrt([+int|+ilist],-ilist))?

% Declaration des modes pour les litteraux en corps de clause
:- modeb(1,qsrt(+ilist,-ilist))?
:- modeb(1,part(+int,+ilist,-ilist,-ilist))?

```

12. Le système PROGOL est accessible par ftp à l’adresse suivante: <ftp://ftp.comlab.ox.ac.uk/pub/Packages/ILP>

Algorithme 5.3 Algorithme PROGOL

tant que il reste des exemples positifs **faire**

pour chaque exemple positif e **faire**

Construire la clause c_1 la plus spécifique qui implique l'exemple e

Trouver une clause c_2 plus générale que c_1 (au sens de la θ -subsumption) telle que la mesure de compression soit maximale

Retirer tous les exemples couverts par la clause c_2

fin pour

fin tant que

```

:- modeb(1,append(+ilist,[+int|+ilist],-ilist))?

% Types
ilist([]).
ilist([Head|Tail]) :- int(Head), ilist(Tail).

% Theorie du domaine definie en intension
part(X,[],[],[]).
part(X,[X|Tail],List1,List2) :- part(X,Tail,List1,List2).
part(X,[Head|Tail],[Head|Tail1],List2) :-
    Head < X, part(X,Tail,Tail1,List2).
part(X,[Head|Tail],List1,[Head|Tail2]) :-
    Head > X, part(X,Tail,List1,Tail2).

append([],List,List).
append([Head|Tail],List1,[Head|List2]) :- append(Tail,List1,List2).

```

Exemples positifs

```

qsort([],[]).
qsort([3,2,1],[1,2,3]).
qsort([X],[X]).
qsort([X,Y],[X,Y]) :- X < Y.
qsort([Y,X],[X,Y]) :- X < Y.
qsort([X,Y,Z],[X,Y,Z]) :- X<Y, Y<Z.
qsort([X,Z,Y],[X,Y,Z]) :- X<Y, Y<Z.
qsort([X,Z,Y],[X,Y,Z]) :- X<Y, Y<Z.
qsort([Y,X,Z],[X,Y,Z]) :- X<Y, Y<Z.
qsort([Y,Z,X],[X,Y,Z]) :- X<Y, Y<Z.
qsort([Z,X,Y],[X,Y,Z]) :- X<Y, Y<Z.

```

```

:- qsort([0,2,1],[0,2,1]).
:- qsort([0,2,1],[0,1]).
:- qsort([1,0,2],[2,0,1]).
:- qsort([1,0,2],[2,1,0]).
:- qsort([1,2,0],[1,0,2]).
:- qsort([0,2,1],[2,1,0]).
:- qsort([2,1,0],[2,1,0]).
:- qsort([2,0,1],[2,1,0]).
:- qsort([2,1],[1]).
:- qsort([1],[2]).
:- qsort([0,1,2],[1,0,2]).
:- qsort([0,1],[1,0,1]).

```

Exemples négatifs

Le système PROGOL induit une définition du prédictat *qsort*/2 sous la forme des clauses suivantes :

```

qsort([],[]).
qsort([A|B],C) :- qsort(B,D), part(A,D,E,F), append(E,[A|F],C).

```

Sur cet exemple, la stratégie de parcours de l'espace de recherche par PROGOL est bien mise en évidence. La clause la plus spécifique est d'abord calculée, puis PROGOL part de la clause la plus générale et la spécialise en ajoutant des littéraux. Pour chaque clause examinée, PROGOL compte le nombre d'exemples positifs et négatifs couverts et fait une mesure de compression qui lui permet de sélectionner la clause à retenir.

5.6 Les domaines d'application de la PLI

Découpe en éléments finis

La découpe en éléments finis est une technique très utilisée par les ingénieurs qui cherchent à analyser des contraintes sur des structures physiques [DM92]. La figure 5.8 est un exemple de découpage en éléments finis. Les contraintes qui s'exercent sur la structure peuvent être exprimées par des équations différentielles. Cependant il n'est pas possible de résoudre de telles équations à l'aide d'un ordinateur en un temps acceptable. Pour résoudre ce problème, les ingénieurs découpent la structure en un nombre fini d'éléments, et utilisent des approximations linéaires pour calculer les contraintes sur chaque partie élémentaire de la structure.

Dans la pratique, il n'existe pas de méthode générale permettant de réaliser un maillage acceptable pour une pièce. GOLEM a été utilisé dans ce contexte en vue d'induire un modèle de la méthode utilisée par les experts pour découper les structures. Il faut noter que les experts ne sont pas capables de modéliser leur méthode de découpage et que GOLEM est ici utilisé pour extraire cette expertise.

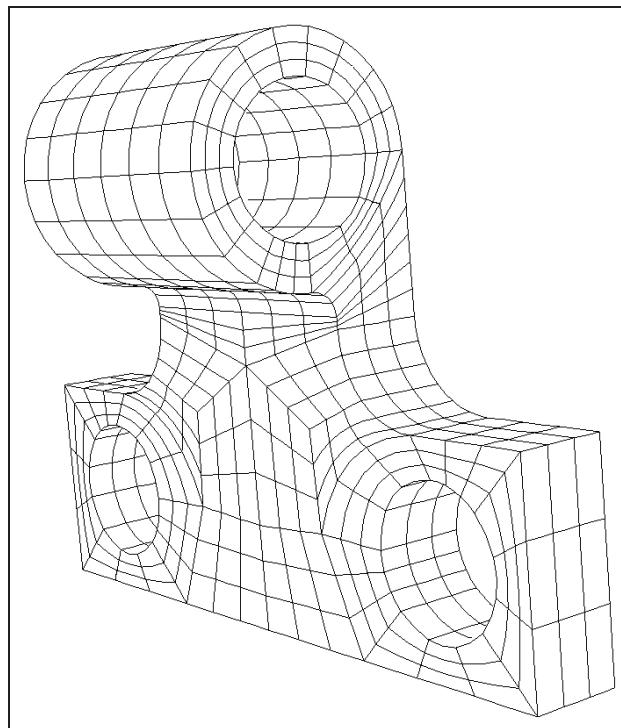


FIG. 5.8 – Une pièce découpée en parties élémentaires.

La densité du maillage dépend des propriétés géométriques de l'objet, des forces et des contraintes qui agissent sur l'objet, et des relations entre les différents composants de l'objet.

Pour traiter ce problème avec la PLI, une structure est représentée par un ensemble d'arêtes, des propriétés sur les arêtes, et des relations entre les arêtes. Chaque objet est ainsi décrit en spécifiant :

- Le type de ses arêtes : longue importante, importante, courte importante, pas importante, circuit, demi-circuit ...
- Les différentes conditions existants aux zones communes à plusieurs arêtes : libre, fixée sur un côté, fixée sur deux côtés, complètement fixée.
- Les contraintes de charge qui s'exercent sur chaque arête : pas de charge, chargée sur un côté, chargée sur les deux côtés, chargée de manière continue.
- La représentation géométrique de l'objet. Les auteurs s'attachent à la notion d'arêtes voisines, d'arêtes opposées et d'arêtes identiques. On spécifie que les arêtes sont opposées si elles jouent un rôle symétrique dans la pièce. Certaines arêtes sont non seulement opposées, mais elles ont de plus la même forme et la même longueur. On précise alors qu'elles sont identiques.

Pour chaque arête a_i d'une structure s , on dispose d'un fait

$$\text{mesh}(a_i, n)$$

qui signifie que cette arête est découpée en n éléments finis.

Le prédicat à apprendre est donc $\text{mesh}(\text{Arete}, N)$, où Arete est une arête et N est le nombre de segments qui constitueront cette arête. Avec des descriptions de trois objets, GOLEM induit cinquante-six règles différentes. Ces règles ne sont pas toutes pertinentes, mais GOLEM induit des règles de la forme :

```
mesh(A, 1) :-  
    not_important(A),  
    not_loaded(A).
```

qui signifie qu'une arête A est découpée en un élément si elle n'est pas importante et n'a pas de contrainte.

Une autre règle induite est :

```
mesh(A, B) :-  
    cont_loaded(A),  
    same(A,C),  
    cont_loaded(C),  
    mesh(C, B).
```

qui met en évidence que si deux arêtes sont identiques et qu'elles sont contraintes de manière continue, leur découpage se fait également en un nombre identique d'éléments.

Il est intéressant de noter que selon les experts du domaine, les règles fournies par le système mettent en évidence des dépendances qui leur étaient jusqu'alors inconnues [BM95b].

Une liste d'applications

Le réseau d'excellence européen en PLI (ILPNet) a rassemblé sur son site <http://www-ai.ijs.si/ilpnet2/apps/index.html> une liste de liens vers des descriptions d'applications où l'ILP est utilisée pour l'apprentissage de concepts relationnels. En voici une liste partielle :

Sciences de la vie : structure 3-D des protéines, découverte de neuropeptides, analyse de tests de laboratoire, diagnostic précoce de rhumatisme, mutagénèse.

Environnement : développement des algues, données de qualité de l'eau.

Langage naturel : règles d'accord des verbes au passé en anglais, apprentissage d'analyseurs syntaxiques, définition des groupes nominaux, catégorisation de texte, analyse de groupes syntaxiques, prédiction des césures, etc.

Autres domaines : bases de données géographiques, métallurgie, analyse des accidents de la circulation, marketing et commerce.

Pour ne prendre que l'exemple du langage naturel, les techniques d'apprentissage semblent donc être tout à fait adaptées pour inférer un analyseur morphologique, syntaxique ou sémantique d'un langage. De plus, Prolog est un langage considéré comme bien adapté au traitement du langage naturel, ce qui fait également de la programmation logique inductive une méthode appropriée pour cette tâche.

Les premiers travaux dans le domaine ont été réalisés par J. Zelle et R. Mooney [ZM93] [Zel95] [Moo96]. Ils se poursuivent en particulier dans les domaines cités ci-dessus.

5.7 Les chantiers de la PLI

5.7.1 Les problèmes à résoudre

5.7.1.1 Une transition de phase rédhibitoire?

L'attrait majeur de la PLI est le recours à un langage des hypothèses expressif (plus que la plupart des autres langages de généralisation) et commode, puisqu'il permet la représentation aussi bien des exemples que des hypothèses ou que la théorie du domaine. La logique des prédictats est de plus un domaine bien connu grâce aux travaux des logiciens. Comme nous l'avons vu, utiliser la PLI nécessite cependant de choisir avec soin les relations d'ordre et les opérateurs d'exploration. Imposer des limites pour obtenir des ordres bien fondés et des systèmes d'induction contrôlés introduit des difficultés, certes à prendre au sérieux, mais semble-t-il non rédhibitoires... Mais des travaux récents mettent en cause la possibilité même de l'induction supervisée lorsque le langage d'hypothèses est trop expressif. Nous allons examiner ce nouveau problème.

Ces travaux sont fondés sur des considérations complètement indépendantes de celles que nous avons développées dans le chapitre 2 qui dérivent de l'analyse statistique de l'apprentissage. Ils portent une objection sérieuse à l'encontre de la possibilité de la PLI, et plus généralement de l'induction supervisée avec des langages expressifs.

Lorsqu'on fait de l'apprentissage supervisé, on se guide dans l'espace des hypothèses en mesurant l'adéquation des hypothèses testées par rapport aux exemples d'apprentissage, selon le principe *ERM*. On cherche donc les hypothèses dont le risque empirique est le plus faible. Souvent, on utilise la fonction de perte $\{0, 1\}$ qui compte le nombre d'exemples positifs non couverts et le nombre d'exemples négatifs couverts indûment. La recherche dans \mathcal{H} s'effectue par une sorte de gradient, en cherchant quelles modifications des hypothèses courantes pourraient conduire à des hypothèses de meilleur risque empirique (voir le chapitre 3).

Il est intéressant de connaître le paysage du risque empirique en fonction des hypothèses $h \in \mathcal{H}$. Celui-ci dépend de l'échantillon d'apprentissage et en particulier de l'étiquetage des exemples. Si on veut tirer des conclusions générales, il faut cependant faire des expériences ne prenant pas en compte cet étiquetage. Il est clair que le risque dépend directement de la couverture des exemples par les hypothèses, et donc qu'une hypothèse tellement générale qu'elle couvre pratiquement tous les exemples ne pourra être bonne que si le concept cible est très général, et inversement pour des hypothèses tellement spécifiques qu'elles en couvrent pratiquement aucun

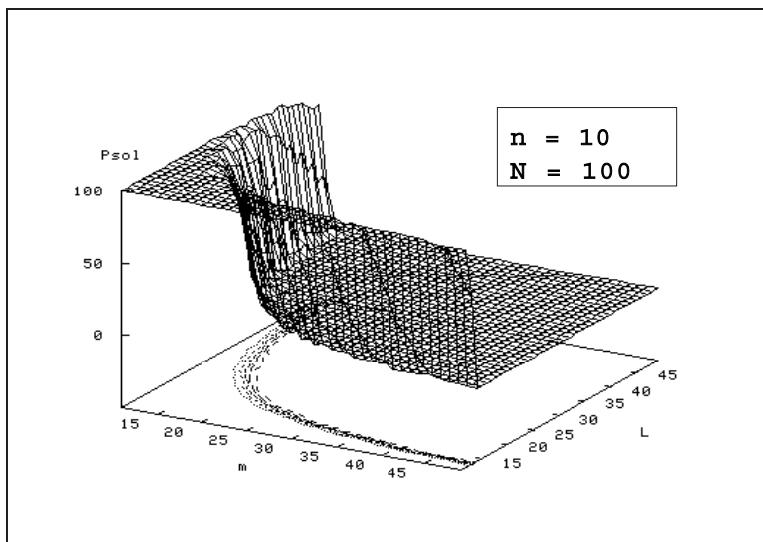


FIG. 5.9 – Une courbe tridimensionnelle de la probabilité de couverture d'un exemple tiré aléatoirement par une hypothèse h en fonction de m , le nombre de littéraux dans h et de L , le nombre d'atomes clos (ou constantes). m et N correspondent respectivement au nombre de variables (10 dans cette expérience) et au nombre de tuples dans les relations (100 ici). Les hypothèses les plus générales se situent à gauche sur cette figure, correspondant à des hypothèses peu contraintes (m et L petits).

exemple. Il est donc intéressant d'examiner le taux de couverture¹³ des hypothèses de \mathcal{H} . Cela permettra en particulier d'analyser la relation entre la variation syntaxique des hypothèses dans leur langage $\mathcal{L}_{\mathcal{H}}$ et leur taux de couverture : deux hypothèses proches dans $\mathcal{L}_{\mathcal{H}}$ ont-elles des taux de couverture proches, et donc un risque empirique proche ?

C'est ce qu'ont étudié Giordana et Saitta ([GS00]) dans le cas d'espaces d'hypothèses composés de clauses de Horn. Par exemple, en faisant varier le nombre de variables, le nombre de littéraux et le nombre d'atomes clos (et en répétant ces expériences un grand nombre de fois), ils ont obtenus des courbes telles que celle de la figure 5.9 qui rappellent des phénomènes de *transition de phase* en physique. De manière surprenante, le taux de couverture mesuré sur les hypothèses y passe brutalement de 0 à 1 sans presque de transition lorsque l'on va progressivement vers des hypothèses plus générales. Ces courbes sont étonnantes. Pourquoi ?

Concrètement, imaginons un algorithme d'apprentissage ascendant, opérant par généralisation progressive et prudente. Lorsqu'une hypothèse courante ne couvre pas tous les exemples positifs connus, le système tente de la généraliser un peu afin d'en couvrir davantage, tout en essayant de ne pas couvrir les exemples négatifs. Le nombre de généralisations possibles est en général très grand et le système se guide en choisissant celles qui conduisent à des hypothèses de meilleur risque empirique. Or que se passe-t-il selon les courbes de taux de couverture observées ? Lorsque l'on prend une hypothèse trop spécifique, elle ne couvre pratiquement aucun exemple, positif ou négatif, son taux de couverture est nul. Le problème, c'est que quand on la généralise, son taux de couverture reste nul. Le risque empirique reste donc constant et il est impossible de se guider dans l'espace de recherche. Ce n'est que lorsque les hypothèses considérées ont été suffisamment généralisées qu'elles se trouvent dans la région de la transition de phase (la

13. C'est-à-dire le pourcentage d'exemples couverts par une hypothèse. Si le taux est de 100 %, c'est que l'hypothèse couvre tous les exemples. Elle ne peut donc pas distinguer des exemples négatifs.

« falaise ») que leur taux de couverture varie et que l'on peut enfin comparer les hypothèses entre elles. Avant, il est impossible au système de se guider. La recherche devient donc aléatoire. Le même phénomène se produit pour un système descendant qui spécialise progressivement des hypothèses initialement trop générales.

L'induction supervisée par généralisation ou spécialisation progressive semble donc selon cette analyse très difficile, sauf pour des problèmes « jouet ». C'est d'ailleurs ce que semblent confirmer empiriquement des expériences complémentaires de Giordana et Saitta. Il y a là un problème fondamental qui relève encore de la recherche théorique et expérimentale. Il touche en tout cas les systèmes utilisant la logique des prédictats comme langage de description des hypothèses. On ne sait pas encore dans quelle mesure il se produit sur d'autres représentations des connaissances, par exemple en inférence grammaticale (voir le chapitre suivant).

5.7.1.2 Quelques problèmes ouverts en PLI

L'extension de la PLI repose en particulier sur la résolution des problèmes suivants :

- la prise en compte de données numériques ;
- l'utilisation d'autres logiques (confirmatoires, modales ...) ;
- l'apprentissage effectif de règles récursives ;
- la prise en compte de l'incertain.

Notes historiques et sources bibliographiques

Quoique les fondations de la logique remontent (facilement) à Aristote, c'est seulement durant le *XIX^e* et le *XX^e* siècles que des théoriciens tels que Boole et Frege l'ont formalisée rigoureusement dans un cadre mathématique. Au *XX^e* siècle, une école de philosophes connus sous le nom de positivistes logiques a promu l'idée que la logique était le socle ultime non seulement des mathématiques, mais de toute la science. Selon ce point de vue, toute affirmation mathématique peut être exprimée dans le cadre de la logique des prédictats, et tous les raisonnements scientifiques valides sont basés sur des dérivations logiques à partir d'un ensemble d'axiomes. Le positivisme logique trouva un support puissant dans les travaux de Gödel dans les années 1930 qui ont montré qu'un ensemble réduit de règles de dérivation était suffisant (complet) pour dériver toutes les conséquences de formules en logique des prédictats. Plus tard, Robinson, en 1965, démontra que l'inférence déductive en logique des prédictats pouvait être réalisée à l'aide d'une unique règle d'inférence, la résolution. Pour appliquer le principe de résolution, il faut mettre les formules logiques sous la forme de clauses logiques. La découverte de Robinson a été d'une importance fondamentale pour l'automatisation de la déduction. Colmerauer et Kowalski furent les pionniers au début des années 1970 du développement de Prolog, un langage de programmation logique. Dans ce langage, toutes les formules logiques sont décrites sous la forme de clauses de Horn.

C'est Plotkin qui dans sa thèse en 1971 [Plo71a] a posé les fondations de ce qui allait devenir le champ de la programmation logique inductive. Plotkin ne se limitait pas à la logique des clauses de Horn, ce qui n'est pas surprenant puisque la programmation logique et le langage Prolog n'existaient pas encore. Ses principales contributions concernent l'introduction de la subsomption relative comme relation de généralité entre clauses, et un algorithme de calcul du moindre généralisé d'un ensemble de clauses sans théorie du domaine, ce généralisé étant la borne inférieure (unique en l'absence de théorie du domaine) de l'espace des généralisations d'un ensemble de clauses. Plotkin a démontré que malheureusement, il n'existe pas en général de

moindre généralisé fini de deux clauses en présence d'une théorie du domaine. Cela restreignait évidemment la portée de son algorithme et c'est pourquoi Shapiro [Sha83] étudia une autre approche, portant sur l'induction descendante de clauses de Horn, du général au spécifique, au lieu d'ascendante comme pour Plotkin. De plus, Shapiro étudia le débogage algorithmique de programmes Prolog par identification automatique de la clause défective. Ces systèmes d'induction étaient cependant très inefficaces et limités à de tout petits problèmes.

Sammut et Banerji, (voir [Sam93]) ont décrit un système appelé MARVIN qui généralise un seul exemple à la fois par rapport à un ensemble de clauses inscrites dans la connaissance préalable ou théorie du domaine (*background knowledge*). À chaque étape, un ensemble d'atomes clos F représentant l'exemple sont appariés avec le corps d'une clause de la théorie du domaine $H \leftarrow B$. Les faits unifiés $B\theta$ sont remplacés par $H\theta$, où θ est la substitution `xxxground`. Il s'agit du seul opérateur de généralisation utilisé. Le système a été testé sur un ensemble de problèmes variés. Plus tard, Muggleton et Buntine démontrèrent que l'opérateur de généralisation de Banerji était un cas particulier de l'inversion d'un pas dans la preuve par résolution. Un ensemble de contraintes plus générales a été fourni et les deux opérateurs en 'V' ont été proposés. L'autre paire d'opérateurs en 'W', également basés sur l'inversion de la résolution, est également issue de leurs travaux. D'autres travaux ont aussi porté sur l'invention de prédicats, mais pour le moment ils sont restés confinés à la logique des propositions et non à la logique des prédicats.

Plus récemment Quinlan, [Qui90], a proposé le système FOIL qui induit des clauses de Horn de manière très efficace. La méthode utilise une heuristique de recherche descendante, par spécialisation progressive, guidée par un critère de gain d'information (voir le chapitre 11). L'un des inconvénients de la méthode est qu'elle est gloutonne et donc susceptible de tomber dans des optima locaux mauvais, voire d'errer sans direction comme le montre le phénomène de transition de phase (évoqué dans la section 5.6).

Ce chapitre est en partie extrait des notes de cours de Marc Bernard, de l'université de Saint-Etienne [Ber00], qui nous a aimablement autorisé à les utiliser. Céline Rouveiro, du LRI (université d'Orsay), a largement contribué à sa rédaction.

Résumé

- La programmation logique inductive (PLI) est essentiellement tournée vers l'apprentissage supervisé de concepts ou de théories (exprimés sous forme de programmes logiques). L'apprentissage y est guidé par une relation de généralité dans l'espace des hypothèses en cherchant des hypothèses cohérentes avec les exemples d'apprentissage.
- Contrairement à la plupart des techniques d'apprentissage qui sont concernées par des connaissances en logique attribut-valeur, la PLI s'occupe de l'apprentissage de règles exprimées en logique des prédictats. Cela implique des choix et des précautions pour la définition d'une relation de généralité entre hypothèses. Les relations principalement étudiées sont la θ -subsumption dont la traduction opérationnelle est la SLD-subsumption qui débouche sur des opérateurs de généralisation par inversion de la résolution.
- Face à la complexité des espaces d'exemples et d'hypothèses en logique des prédictats, il est nécessaire d'utiliser des biais pour contrôler la recherche dans l'espace d'hypothèses.
- Les systèmes d'apprentissage de concepts en PLI se divisent en systèmes descendants, opérant par spécialisations successives afin de ne pas couvrir les exemples négatifs, et les systèmes ascendants opérant par généralisations successives à partir des exemples positifs.
- Il reste beaucoup à faire pour étendre le champ d'application de ces systèmes, en particulier lorsque les données sont numériques et bruitées. Par ailleurs, il faudra aussi résoudre le problème posé par le phénomène de transition de phase en logique des prédictats.

Chapitre 6

Reformulation et transfert de connaissances

Le fil directeur de cet ouvrage est que l'apprentissage implique une exploration de l'espace des hypothèses pour rendre compte des données d'apprentissage. En dehors même des problèmes soulevés dans les chapitres théoriques sur la légitimité de tout critère inductif, cette exploration est généralement intrinsèquement très coûteuse et doit donc être guidée. Lorsqu'une relation de généralité peut être définie sur l'espace des hypothèses, la recherche d'une ou plusieurs hypothèse(s) correcte(s) est considérablement aidée. Cependant, même l'utilisation d'une relation de généralité dans \mathcal{H} peut être insuffisante pour pouvoir résoudre effectivement un problème d'apprentissage. L'espace des hypothèses est en effet généralement extrêmement complexe et riche, et même une recherche informée peut se révéler inefficace. On aimerait alors pouvoir débarrasser le langage de description des hypothèses de tous les descripteurs et relations non pertinents pour la tâche d'apprentissage. On aimerait également pouvoir hiérarchiser la recherche et ne prendre en compte que graduellement les détails de l'univers étudié. De même, il serait pratique de pouvoir bénéficier des explorations passées dans \mathcal{H} à propos de problèmes analogues, voire de pouvoir transcrire des explorations faites dans d'autres espaces d'hypothèses relatifs à d'autres environnements ou univers de connaissances. Finalement, une question essentielle est celle de la construction d'un nouveau domaine de connaissances. Est-il possible d'envisager de le réaliser en s'aidant d'autres domaines ? Est-il envisageable de faire autrement ? Quels mécanismes pourraient être mis en jeu ?

Ce chapitre étudie donc la modification et la construction des espaces d'hypothèses. Les approches pour résoudre ces problèmes sont encore à un stade exploratoire. Elles sont très inspirées de l'étude de la cognition humaine. Il est hors de doute que l'importance du sujet et l'état encore préliminaire de nos connaissances justifieront de nombreuses recherches à l'avenir. C'est cette conviction qui nous a fait écrire ce bref chapitre sur ces questions, même si les travaux actuels sont rares, et les applications encore davantage.

6.1 L'apprentissage en présence de théorie

Jusqu'à présent, nous avons décrit l'apprentissage comme la recherche d'une hypothèse dans \mathcal{H} en bonne adéquation, selon une mesure de risque, avec les données expérimentales décrites dans \mathcal{X} . L'apprentissage peut cependant prendre d'autres visages. Par exemple, un apprenti joueur d'échec humain sera susceptible d'apprendre le concept de « fourchette » (une menace portant simultanément sur plusieurs pièces) à partir de très peu d'exemples, parfois même à partir d'un seul. Comment est-il alors possible de généraliser cette expérience très limitée à toutes les situations de fourchettes, ou au moins à une grande partie d'entre elles ? L'apprentissage peut aussi porter non pas sur la détermination d'une bonne hypothèse mais sur l'efficacité de la recherche de bonnes hypothèses. Il se peut aussi que l'apprenant possède des descriptions de concepts qui soient correctes, mais qui ne soient pas vraiment opérationnelles dans le monde¹.

Le point commun à ces types d'apprentissages est l'intervention d'une connaissance importante sur le domaine d'application, connaissance que l'on appelle « théorie du domaine »². C'est cette connaissance qui permet de pouvoir tirer parti de très peu d'exemples, grâce à des raisonnements enrichissant l'expérience. C'est aussi cette connaissance qui peut éventuellement être modifiée dans l'apprentissage.

6.2 L'apprentissage par examen de preuve (*EBL*)

En schématisant, le problème étudié dans l'apprentissage par examen de preuve (*Explanation-Based Learning : EBL*) est de trouver la réponse associée à une situation ou à une forme $x \in \mathcal{X}$. Il se peut que la recherche de cette réponse soit impossible pour le système, ou du moins qu'elle soit très coûteuse. L'apprentissage consiste alors à rendre la recherche plus facile la prochaine fois, et ceci non seulement pour la situation x , mais aussi pour des situations semblables. Il faut bien sûr définir la notion de similarité. Dans le cas de l'*EBL*, elle est obtenue à partir de l'utilisation de la théorie du domaine.

6.2.1 Le principe de l'*EBL*

On suppose que le système est soumis au problème $x \in \mathcal{X}$, qui peut aussi bien être de trouver une étiquette dans une tâche de classification, comme de trouver un plan d'action. Le système cherche alors une solution $h(x)$ à ce problème. Si la recherche de cette solution est difficile, voire irréalisable, il peut être intéressant pour le système d'examiner la solution finalement obtenue, par le système ou grâce à un professeur, pour voir dans quelles conditions cette solution (éventuellement généralisée) pourrait être réutilisée dans une situation x' semblable. Conceptuellement, il s'agit d'identifier une région de l'espace des problèmes \mathcal{X} pour laquelle la solution h , ou une solution voisine, est adéquate. La figure 6.1 illustre ce principe. D'une certaine manière, l'apprentissage *EBL* peut être considéré comme une technique d'extrapolation dans l'espace \mathcal{X} guidée par une théorie du domaine. La partie *Apprentissage par approximation et interpolation* (en particulier le chapitre 14) de cet ouvrage explore d'autres techniques d'extrapolation fondées

-
1. Un exemple, sans doute extrême, est celui d'un système de diagnostic de panne d'ordinateur capable de caractériser les pannes à partir des équations de Shrödinger associées aux atomes de l'appareil. Cette connaissance même correcte, ne serait cependant sans doute pas d'une grande utilité pratique et la recherche d'une description plus opérationnelle pourrait légitimement être considérée comme un apprentissage à part entière.
 2. Cette « théorie » peut prendre la forme très formelle de programmes en logique du premier ordre, voir le chapitre 5, comme la forme de réseaux sémantiques souvent liés à des types d'inférence non déductifs: propagation de marqueurs par exemple.

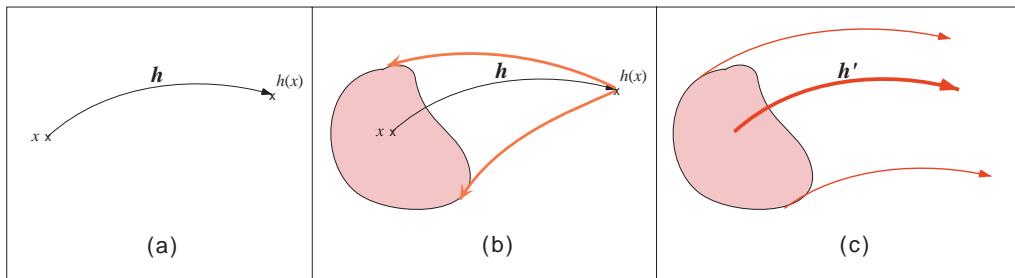


FIG. 6.1 – (a) À partir d'une solution h , trouvée ou fournie, pour le problème x , l'apprentissage EBL cherche sous quelles conditions la solution h est applicable. (b) Ces conditions suffisantes sont obtenues par un mécanisme de régression à travers la théorie \mathcal{T} du domaine. (c) Cela permet de trouver à la fois une généralisation de la solution initiale h et une région de l'espace des problèmes \mathcal{X} pour laquelle cette solution généralisée est applicable.

sur des mesures numériques et des propriétés de lissage. Peu d'auteurs ont examiné le lien entre les deux approches (une exception est [DF97a])

La solution généralisée peut prendre trois formes :

1. Celle d'une *solution toute faite*, appropriée pour une classe de problèmes dans \mathcal{X} , ou bien, comme c'est souvent le cas, pour la reconnaissance de concept. Il s'agit alors d'apprendre une définition opérationnelle de concepts.
2. Celle de *macro-opérateurs*, c'est-à-dire d'étapes de solution. Cela peut permettre d'accélérer la recherche d'une solution en diminuant la profondeur de recherche nécessaire.
3. Celle de *règles de contrôle* ou *heuristiques*, permettant de guider la recherche d'une solution dans l'arbre des possibilités.

Évidemment, ces trois types de connaissances apprises ne sont pas exclusifs l'un de l'autre, mais dans la pratique, les concepteurs de systèmes d'apprentissage par EBL ont privilégié l'une ou l'autre de ces possibilités. Pour des fins d'illustration dans la suite de cette section, nous avons choisi de montrer seulement comment l'apprentissage peut déboucher sur la première voie.

6.2.2 Une illustration de l'apprentissage EBL

Nous montrons dans cette section comment l'apprentissage EBL permet d'apprendre une définition opérationnelle et générale d'un concept à partir d'un exemple positif unique. Pour cela nous reprenons l'exemple devenu célèbre de l'apprentissage du concept d'objets empilables, initialement exposé dans [MKKC86] qui est aussi l'article fondateur de l'EBL.

Soit un monde d'objets caractérisés par les attributs : *couleur*, *volume*, *propriétaire*, *matériau*, *type* et *densité*. Il existe de plus une relation possible entre les objets : la relation *sur* traduisant qu'un objet est sur un autre. La tâche consiste à apprendre le concept d'empilement d'un objet sur un autre. Plus précisément, il s'agit d'apprendre une définition du prédicat :

`empilable(Objet1,Objet2)`

vrai lorsque l'on peut empiler l'`Objet1` sur l'`Objet2`. Nous supposons que l'apprenant dispose préalablement d'une théorie du domaine \mathcal{T} décrite sous la forme de règles logiques que nous décrirons ici en utilisant, comme dans le chapitre 5, la syntaxe Prolog :

```
(T1) : poids(X,W) :- volume(X,V), densité(X,D), W is V*D.
(T2) : poids(X,50) :- est_un(X,table).
```

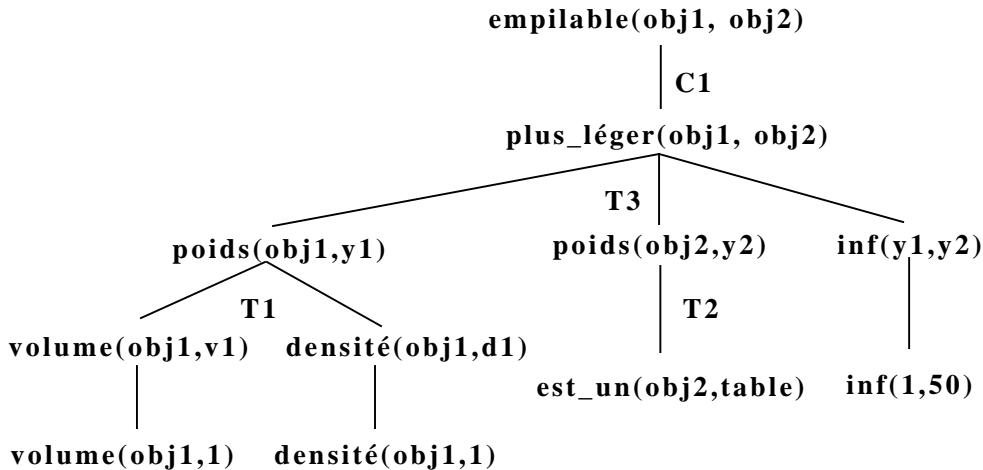


FIG. 6.2 – Un arbre de preuve obtenu dans la théorie du domaine \mathcal{T} montrant que l'exemple satisfait bien le concept cible.

(T3) : $\text{plus_léger}(X, Y) :- \text{poids}(X, W_1), \text{poids}(Y, W_2), W_1 < W_2$.

On suppose de plus que l'apprenant cherche une définition opérationnelle (c'est-à-dire facilement évaluable) du *concept but*:

(C1) : $\text{empilable}(X, Y) :- \text{plus_léger}(X, Y)$.

(C2) : $\text{empilable}(X, Y) :- \text{not}(\text{fragile}(Y))$.

Plus formellement, le critère d'opérationnalité impose que le concept soit exprimé à l'aide de prédictats utilisés pour décrire l'exemple (*volume*, *densité*, *couleur*,...) et de prédictats facilement évaluables (comme le prédictat $<$).

Finalement, l'apprenant est pourvu de l'*exemple positif* décrit par les faits suivants :

```

sur(obj1,obj2).
est_un(obj1,boite).
est_un(obj2,table).
couleur(obj1,rouge).
couleur(obj2,bleu).
volume(obj1,1).
volume(obj2,0.1).
proprietaire(obj1,frederic).
proprietaire(obj2,marc).
densite(obj1,0.3).
materialu(obj1,carton).
materialu(obj2,bois).
  
```

L'apprentissage *EBL* consiste d'abord à examiner une preuve, exprimée à l'aide de la théorie du domaine, que l'exemple fourni est bien un exemple positif du concept cible, puis à généraliser cette preuve pour en extraire un ensemble de contraintes suffisant à garantir la validité de la preuve générale.

Le *premier pas* nécessite que la théorie du domaine soit suffisante pour qu'il soit possible de trouver une preuve, aussi appelée explication (d'où l'appellation *EBL*), montrant que l'exemple satisfait au concept cible. Chaque branche de la preuve peut éventuellement être réduite pour se terminer sur une expression vérifiant le critère d'opérationnalité. Dans le cas présent, l'explication de l'exemple est donnée sous forme graphique dans la figure 6.2.

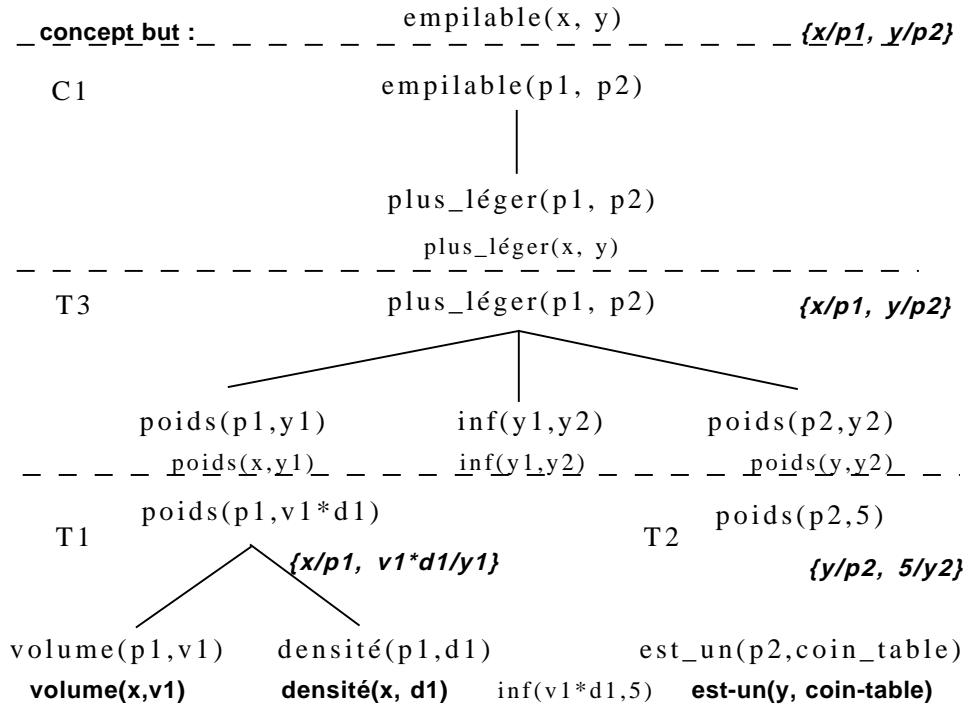


FIG. 6.3 – L’arbre de preuve généralisé est obtenu par régression du concept cible dans l’arbre de preuve en calculant à chaque étape (dénotée par les lignes en pointillés) les littéraux les plus généraux permettant cette étape (indiqués en gras). À la fin de ce processus, la conjonction des littéraux obtenus aux feuilles de l’arbre généralisé fournit la définition opérationnelle et généralisée du concept cible.

Le *second pas* consiste à analyser l’arbre de preuve trouvé pour identifier les conditions de son applicabilité afin de découvrir l’ensemble des situations dans lesquelles il pourrait s’appliquer. Pour le cas décrit ici, la figure 6.2 décrit la généralisation de la preuve donnée dans la figure 6.3.

L’exemple positif initial, le seul connu pour le concept cible, a été généralisé de plusieurs points de vue. D’abord, il était décrit par une collection d’attributs dont beaucoup, n’intervenant pas dans la preuve, se révèlent non pertinents et peuvent donc être éliminés. Ensuite, de nombreuses constantes ont été remplacées par des variables. Notons cependant que ce n’est pas systématique. La constante numérique 50, par exemple, demeure dans la preuve généralisée car elle n’a pas été introduite par une instanciation aux feuilles de l’arbre de preuve mais par une règle de la théorie. Finalement, la généralisation va plus loin que cette simple variabilisation grâce à la *procédure de régression* d’une formule à travers une règle proposée par Waldinger [Wal77]. Cette procédure calcule les conditions suffisantes d’une conclusion C en fonction d’une preuve P , c’est-à-dire un ensemble d’assertions A tel que A entraîne C grâce à P . Dans le cas de la régression d’une conclusion à travers un arbre de preuve, la procédure fonctionne itérativement à travers les étapes de la preuve, calculant à chaque fois les conditions suffisantes de la conclusion de l’étape, en fonction de la règle de dérivation employée pour cette étape. La procédure se termine lorsqu’elle a parcouru toutes les étapes de la preuve initiale pour atteindre les feuilles de l’arbre de preuve.

Un exemple de l’application de cette procédure est donné par la figure 6.3. En fonte normale se retrouvent les éléments de la preuve initiale trouvée pour l’exemple positif du concept `empilable(Obj1, Obj2)`. La frontière de chaque étape de régression est indiquée par les lignes en

pointillés, avec à chaque fois en gras les conditions suffisantes calculées. Par exemple, le concept cible général est `empilable(X,Y)`. On calcule sa régression à travers la règle `empilable(X,Y) :- plus_léger(X,Y)`, ce qui produit la précondition `plus_léger(X,Y)`. On continue alors en calculant la régression de cette expression à travers (T3) le pas suivant de la preuve, ce qui produit à l'ensemble de préconditions `{poids(X,Y1), inf(Y1,Y2), poids(Y,Y2)}`. Étape par étape, on atteint ainsi les feuilles de l'arbre de preuve, produisant la définition générale suivante :

```
empilable(X,Y) :- volume(X,VX), densité(X,DX),
est_un(Y,table), inf(VX,DX,50).
```

Cette procédure a été implantée en particulier dans l'algorithme Prolog-EBG [KCC87]. Lorsqu'une théorie correcte et complète lui est fournie, elle produit une hypothèse (ensemble de règles Prolog) elle-même correcte et couvrant les exemples positifs connus des concepts à apprendre. Ces règles sont des conditions suffisantes décrivant les concepts cible en fonction de la théorie du domaine.

6.2.3 Discussion sur l'apprentissage de concept à partir d'explications

Plusieurs problèmes méritent d'être soulignés.

- Il est important de remarquer que l'algorithme rapidement décrit ci-dessus fournit seulement une définition suffisante (et non nécessaire) du concept cible. Ceci est dû au fait que la preuve est guidée par les exemples fournis qui n'épuisent pas forcément toutes les possibilités de preuve du concept. Par exemple, dans l'apprentissage du concept `empilable(X,Y)`, la théorie propose deux définitions, l'une se rapportant au fait que `X` est plus léger que `Y`, l'autre au fait que `Y` n'est pas fragile. L'exemple positif fourni ne se rapporte qu'à la première définition, et les conditions trouvées par régression à travers la preuve relative à cet exemple ne concernent donc que la première définition possible du concept. C'est bien pourquoi on parle d'apprentissage à partir d'explications. Si l'on apprend bien une généralisation de l'exemple positif, il s'agit cependant d'une spécialisation de la définition initiale du concept cible.

Des commentateurs de l'apprentissage *EBL* ont fait remarquer que d'un certain côté l'exemple positif fourni est redondant et inutile. On pourrait très bien apprendre une définition complète du concept cible par régression selon tous les arbres de preuve possibles en arrêtant les preuves lorsque le critère d'opérationnalité est satisfait comme cela se fait en programmation logique par évaluation partielle [HB88]. L'intérêt des exemples n'est cependant pas négligeable. Ils permettent en effet d'orienter la recherche de preuve dans un espace de possibilités généralement immense. De plus, si ces exemples sont bien choisis, ils sont représentatifs de l'environnement et guident donc l'apprentissage vers les seules régions de \mathcal{X} pertinentes. Ainsi, tant l'apprentissage qu'ensuite la reconnaissance d'un concept sont accélérés par élimination des cas de preuves inutiles (c'est d'ailleurs l'une des raisons pour laquelle les Anglo-saxons parlent de *speed-up learning*).

- L'exemple d'apprentissage ci-dessus se traitait à l'aide d'une théorie du domaine à la fois correcte, complète (toute formule vraie dans l'interprétation intentionnelle de la théorie doit être déductible de la théorie), et adéquate, c'est-à-dire décrite dans des termes pouvant conduire à une reformulation utile des concepts. Il est évident que la qualité des descriptions apprises par *EBL* dépend grandement des propriétés de la théorie du domaine. C'est là l'une des plus importantes limitations pour la mise en pratique de la méthode car il est

rare que l'on dispose d'une théorie parfaite avant apprentissage. On peut caractériser les problèmes en trois classes :

1. Le cas d'une *théorie incomplète*. Cela peut avoir pour conséquence l'impossibilité de trouver une preuve pour un exemple positif, ou l'obtention d'une preuve incomplète, voire de multiples preuves mutuellement incohérentes.
2. Le cas d'une *théorie incorrecte*. Dans ce cas, il peut y voir découverte de preuve incorporant des contradictions avec le monde réel, c'est-à-dire apprentissage de mauvais concepts.
3. Finalement, il y a le cas de *théorie impossible à traiter*, interdisant de trouver des preuves de l'exemple fourni.

Ces trois classes de problèmes ont fait l'objet de travaux, surtout la première. C'est ainsi qu'on a cherché à étendre la notion de preuve à des preuves « plausibles » exprimées dans des extensions de la logique (incertaines ou floues). On a également cherché à compenser l'insuffisance de la théorie du domaine par l'utilisation simultanée de plusieurs exemples.

- S'il est intéressant de savoir utiliser des exemples positifs d'un concept à l'aide d'une théorie du domaine, il peut être tout aussi tentant d'utiliser des exemples négatifs. Par exemple, on pourrait vouloir apprendre à partir d'un échec à mater le roi adverse dans une situation pourtant apparemment favorable. Deux approches sont envisageables dans ce cas. La première consiste à traiter l'exemple négatif comme un exemple positif, le généraliser et utiliser la négation du concept appris pour caractériser les situations nécessaires au succès. La seconde part de l'échec d'une preuve d'un exemple négatif pour analyser comment il faudrait la modifier pour parvenir à un succès et ainsi caractériser ce qui sépare les situations positives des situations négatives. L'apprentissage à partir d'échecs (*learning by failure*) a été particulièrement étudié dans le contexte de l'apprentissage de connaissances de contrôle puisqu'il y est crucial de savoir quand il faut éviter d'avoir recours à un opérateur.

6.2.4 L'apprentissage de connaissances de contrôle à partir d'explications

L'apprentissage de concept à partir d'explications requiert donc une théorie du domaine aussi correcte et complète que possible. Un domaine dans lequel il est naturel de supposer une telle théorie est celui de la résolution de problèmes. Dans ce contexte en effet, l'enjeu de l'apprentissage est souvent de rendre efficace un résolveur de problème capable en principe de résoudre n'importe quel problème, mais souvent inutilisable en pratique à cause de la complexité de la recherche d'une solution. L'apprentissage peut alors prendre trois formes :

1. L'apprentissage des conditions dans lesquelles il est intéressant d'envisager l'utilisation d'un opérateur.
2. L'apprentissage de macro-opérateurs qui sont souvent utiles dans un environnement donné et qu'il est donc intéressant de connaître.
3. L'apprentissage d'heuristiques de contrôle permettant de trier par ordre d'intérêt décroissant les opérateurs envisageables dans un état donné.

Pour apprendre à partir d'explications les conditions d'application d'un opérateur, il suffit de le considérer comme un concept dont on cherche à caractériser le domaine d'application. Pour obtenir des exemples positifs d'application d'opérateurs, on examine les séquences d'opérateurs correspondant à des solutions de problèmes. Cela fournit pour chaque opérateur utilisé un ou des état(s) dans le(s)quel(s) il a été employé, et donc un ou plusieurs exemple(s) positif(s) de son application. Un exemple de cette approche figure dans le chapitre 4 avec le système LEX de

Mitchell. On y voyait à l'œuvre un apprentissage par élimination des candidats dans l'espace des versions, une variante appelée LEX2 a été développée, qui utilise un apprentissage à partir d'explications.

La deuxième approche consiste à chercher à améliorer l'efficacité d'un système de résolution de problème ou de planification en définissant des macro-opérateurs correspondant à des sous-séquences généralisées de solutions. Il faut d'une part sélectionner les séquences utiles : si elles sont trop spécifiques, elles serviront rarement et ralentiront inutilement le processus de choix à chaque nœud de la recherche. Inversement, si elles correspondent à des sous-séquences courtes, donc généralement d'un domaine d'application plus vaste, elles seront moins utiles car ne correspondant pas à des « grands pas » dans la solution. Nous reviendrons sur ce problème. D'autre part, il faut apprendre leur domaine d'application, ce qui se fait souvent par régression comme dans l'apprentissage de concept.

La troisième approche est d'apprendre des heuristiques de contrôle guidant le système lorsqu'il cherche une solution à un problème. Un système opérant de cette manière tout en apprenant aussi des macro-opérateurs est le système SOAR [LRN86]. Il utilise pour ce faire un mécanisme appelé *chunking* qui produit des « chunks » (littéralement « gros morceaux ») ou macro-opérateurs. SOAR est un système général de résolution de problème opérant par décomposition de problèmes en sous-problèmes. Grâce à la nature récursive de ce procédé, des hiérarchies de sous-buts sont produites. À chaque fois qu'une impasse est rencontrée dans la résolution d'un problème, un sous-but est engendré et le système cherche à le résoudre par une méthode de recherche faible telle que *générer et tester*³. La solution trouvée pour sortir de l'impasse est alors vue comme une explication de la démarche à suivre pour sortir à l'avenir d'impasses similaires. Le mécanisme de chunking entre en jeu lorsqu'un sous-but a été résolu. Un résumé de la procédure de résolution du sous-but est enregistré sous forme de règles de production. La partie action de chaque règle est fondée sur les résultats du sous-but, la partie condition ou antécédent exprime les aspects pertinents qui déterminent le succès de l'emploi de la procédure. SOAR a été testé dans de très nombreux problèmes et est proposé comme modèle cognitif de résolution de problème [New90].

PRODIGY [Min88, Min90] est un autre système intéressant d'apprentissage d'heuristiques de contrôle. Il s'agit d'un système de planification fonctionnant en utilisant une théorie générale sur la planification, une théorie du domaine et une description des opérateurs à la STRIPS⁴ avec des listes de préconditions, d'ajouts et de retraits exprimées avec des prédicats sur le monde. Comme SOAR, PRODIGY utilise une résolution de problèmes par décomposition en sous-problèmes. Après avoir produit un plan d'action, le système analyse ses bons choix et ses impasses et cherche à les expliquer en termes de sa théorie du domaine. Il est ainsi capable de produire des règles de contrôle telles que :

```

if (and (current-node node)
         (candidate-goal node (on X Y))
         (candidate-goal node (on Y Z)))
then (prefer goal (on Y Z) to (on X Y))

```

Ce genre de règle est obtenu après l'analyse d'un échec lorsque le robot a essayé de placer d'abord la sous-pile d'objets supérieure avant une sous-pile inférieure. Ceci n'est évidemment qu'un petit exemple des capacités de PRODIGY.

-
- 3. Méthodes consistant à engendrer les différents possibilités à chaque point de choix puis à choisir en fonction d'une fonction d'évaluation. La célèbre méthode de recherche A* en est un exemple.
 - 4. STRIPS (Stanford, 1971)[FN71, FN72] est le grand ancêtre de la plupart des systèmes de planification. Il a en particulier fixé la manière de représenter les opérateurs.

Comme SOAR, PRODIGY a été testé intensivement. Il en est ressorti que l'apprentissage de macro-opérateurs et de règles de choix des opérateurs n'était pas nécessairement avantageux. En effet, chaque nouvelle règle, chaque nouvel opérateur ajoute un choix à l'ensemble des choix possibles à chaque étape de résolution. Le coût ainsi encouru peut dépasser l'avantage retiré. C'est ce que Minton, le concepteur principal de PRODIGY a appelé le *problème de l'utilité*. C'est pourquoi PRODIGY incorpore un module de mesure de l'utilisation de chaque règle et opérateur, de l'économie réalisée et du coût en terme de temps de sélection et d'application. Seuls les règles et opérateurs dont l'utilité mesurée est jugée assez élevée sont conservés. De cette manière, les gains empiriquement mesurés sur un ensemble de tâches sont de l'ordre de 20 % à 110 % avec une moyenne autour de 40 %, ce qui sans être négligeable, est toutefois étonnamment faible quand on pense que le fait d'avoir un facteur de branchement (nombre de choix disponibles à chaque noeud) réduit (grâce aux règles de contrôle) devrait permettre un gain exponentiel. Il s'agit là d'un problème qui ressort encore de la recherche.

6.2.5 Bilan sur l'apprentissage à partir d'explications

Dans ce survol de l'apprentissage à partir d'explications, nous avons laissé de côté certains problèmes tels celui de l'apprentissage dans le cas de preuves récursives ou dans le cas de preuves non déductives comme cela se rencontre dans des systèmes à base de schémas. Nous renvoyons le lecteur intéressé à l'abondante littérature portant sur l'apprentissage *EBL*. Il faut surtout retenir que ces méthodes d'apprentissage permettent l'obtention de généralisations justifiées, à partir d'un petit nombre d'exemples, ce qui les distingue radicalement des méthodes inductives classiques. Si, les travaux sur cette approche ont été peu nombreux depuis plus d'une décennie, nous pensons cependant, comme Gerald DeJong l'un de leurs promoteurs, qu'il ne peut s'agir que d'une éclipse passagère. Certes il est difficile de disposer d'une théorie du domaine forte, mais il existe cependant des domaines où de telles théories préexistent, comme la planification, le contrôle ou le diagnostic. Dans ces cas là, et en particulier quand chaque exemple est coûteux à obtenir, l'apprentissage à partir d'explications peut être très intéressant. Plus généralement, on ne voit pas comment l'emploi exclusif de techniques sans usage de théorie du domaine pourrait couvrir l'ensemble des problèmes d'apprentissage. Les techniques telles que l'*EBL* ont donc leurs beaux jours devant elles.

6.3 Abstraction et reformulation des connaissances

Même lorsqu'une approche « nonsymbolique » est revendiquée en intelligence artificielle, les capacités d'inférence et de raisonnement d'un agent dépendent de la manière dont est exprimée sa connaissance et comment elle peut s'interfacer avec la perception de l'environnement. Un pouvoir expressif trop limité de la représentation utilisée peut empêcher certains raisonnements. C'est une des raisons pour lesquelles il faut des millions d'exemples en apprentissage par renforcement pour apprendre le concept de « fourchette » aux échecs, quand quelques exemples suffisent avec une représentation en logique des prédicats et un apprentissage à partir d'explications. D'un autre côté, trop d'informations disponibles peuvent conduire à des raisonnements incapables de trouver une solution en un temps raisonnable. Il est donc crucial pour un agent intelligent de savoir contrôler l'information qu'il prend en compte pour résoudre une tâche. Il serait extrêmement utile d'avoir des systèmes capables d'apprendre ce contrôle. L'état de l'art en apprentissage artificiel est encore loin de pouvoir répondre à ce problème, mais il existe depuis longtemps des travaux de réflexion sur les changements de représentation et les reformulations des connaissances en intelligence artificielle. Nous en évoquons rapidement quelques aspects ici,

renvoyant en particulier le lecteur intéressé à la synthèse très informée de Jean-Daniel Zucker [Zuc01].

L'idée générale des changements de représentation est de trouver une représentation permettant une résolution simple de la tâche courante. Depuis les travaux pionniers d'Amarel en 1968 [Ama68], on sait qu'une bonne représentation peut rendre triviale la recherche d'une solution. J.-D. Zucker distingue trois types de changements de représentation :

- l'*abstraction* : un changement de représentation, dans un même formalisme, qui en cachant des détails et en préservant des propriétés désirables, simplifie la résolution du problème courant ;
- la *reformulation* : une modification du formalisme de représentation tout en conservant les mêmes informations ;
- l'*approximation* : une simplification des inférences, dans la même représentation, afin de simplifier la résolution.

L'abstraction est souvent considérée comme un changement de granularité ou de niveau de détail d'une représentation. Cela néglige le fait qu'il n'y a pas, généralement, de hiérarchie toute faite de niveaux de détails. Il s'agit de « construire » une description de haut niveau pertinente pour le contexte et la tâche. Pour cela plusieurs voies sont possibles. La plus étudiée en intelligence artificielle consiste à considérer le processus d'abstraction comme celui d'une projection d'une représentation dans une autre *modulo* une relation d'équivalence, de telle manière que ne soient distingués dans la nouvelle représentation que les éléments qui doivent l'être pour la résolution du problème courant. Ainsi se trouvent éliminés les « détails » superflus. Une autre approche est de considérer l'abstraction comme un problème pour lequel des opérateurs de changement de représentation sont disponibles et dont on cherche une solution sous forme de séquence de tels opérateurs. La figure 6.4, tirée de [Zuc01] illustre ce point de vue en schématisant la tâche du géographe qui doit représenter sur une carte les éléments d'un paysage photographié par satellite. La carte finallement produite résulte du choix de nombreux opérateurs. On peut alors envisager d'apprendre automatiquement les conditions d'application de chaque opérateur, de manière similaire à ce qui a été discuté dans la section 6.2 sur l'*EBL*.

L'étude de l'abstraction est certainement étroitement liée à celle de l'apprentissage. Il s'agit là d'un vaste champ de recherche qui reste encore largement à explorer.

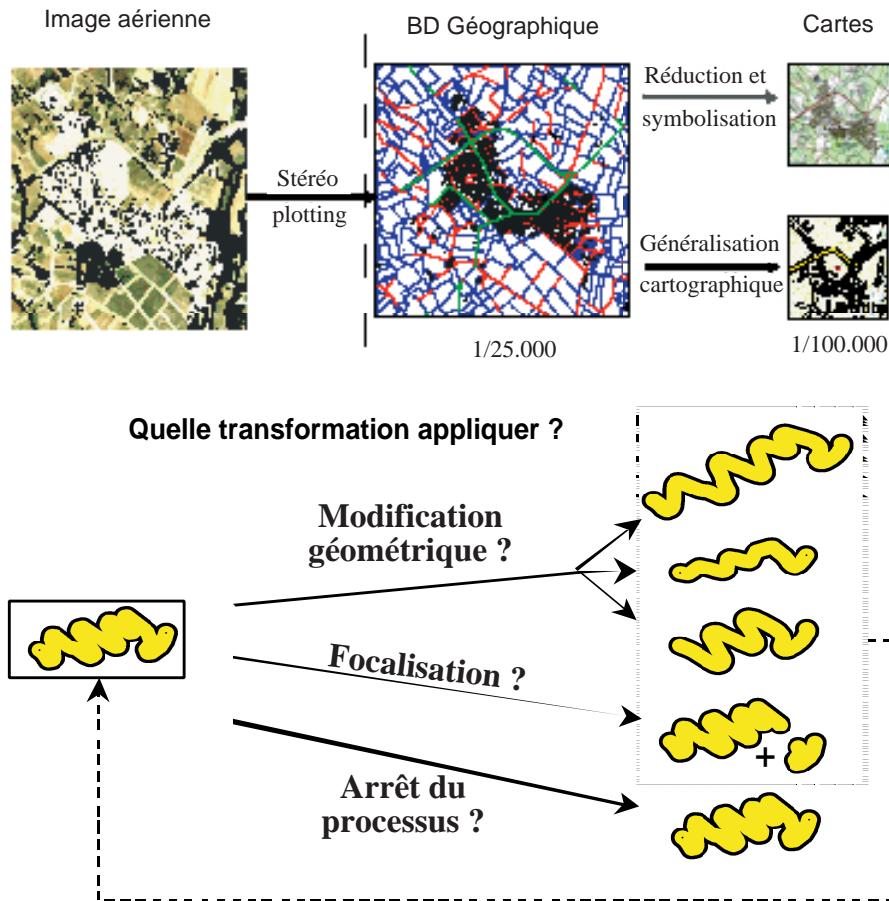


FIG. 6.4 – En haut est schématisé le travail du cartographe : comment traduire une photo aérienne en carte ? En bas, on voit que pour traduire le tracé d'une route par exemple, il peut faire appel à plusieurs opérateurs applicables en séquence. L'une des tâches d'apprentissage consiste à apprendre quand appliquer chacun de ces opérateurs. (Tiré de [Zuc01], p.21).

6.4 Changement de repère et raisonnement par analogie

L'induction suppose connu un échantillon $\mathcal{S} = \{(x_i, f(x_i))\}_{i=1..m}$ à partir duquel on induit une fonction h aussi proche de f que possible permettant la prédiction d'une réponse en tout point de l'espace \mathcal{X} . Le *raisonnement par analogie* suppose beaucoup moins : de la connaissance d'un couple $(x, f(x))$, il cherche à prédire la réponse en un point $x' \neq x$ (voir figure 6.5). Évidemment, en l'absence de plus de données, cela n'est possible que si l'on connaît des propriétés utiles sur l'espace $\mathcal{X} \times \mathcal{F}$.

Dans sa version la plus fruste, le raisonnement par analogie opère comme la méthode par plus proches voisins : lorsqu'il est difficile, voire impossible, de trouver une solution pour le problème $x \in \mathcal{X}$, il peut être intéressant d'emprunter la solution connue d'un problème x' « proche » de x . Toute la question devient alors de caractériser la notion de proximité. Lorsque de plus, le problème x' dont on s'inspire n'appartient pas au même domaine de connaissance, on parlera souvent de similarité. Cette notion est dépendante de la tâche : deux situations peuvent être soit très similaires, soit très différentes en fonction du contexte. Un caillou et mon Psion peuvent tous

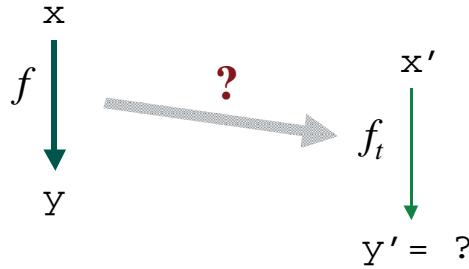


FIG. 6.5 – Schématiquement, l'analogie consiste à trouver la réponse à une question x' connaissant un couple $(x, f(x))$. La fonction produisant la réponse peut être différente en x' et en x .

les deux servir de presse-papier pour empêcher des feuilles de s'envoler pendant que je téléphone à mon coauteur. Pour cette tâche, ils sont similaires. Je me garderai cependant bien d'utiliser l'un et l'autre indifféremment lorsque je veux prendre des notes dans un séminaire.

Nous verrons au chapitre 15 comment se traduit la notion de proximité dans un espace décrit par des attributs numériques. Vapnik [Vap95] parle de *transduction* lorsque l'on cherche à prédire la valeur en un point x' d'une fonction inconnue, sans chercher à déterminer celle-ci directement par induction. Il suppose cependant l'existence d'une telle fonction s'appliquant uniformément dans \mathcal{X} . Cette vision ne prend pas en compte la possibilité qu'il ne faut pas chercher une fonction unique s'appliquant dans tout l'espace des situations, mais qu'il y ait pour chaque situation une fonction définie localement, et qu'il faut « déformer » (minimalemen) pour pouvoir l'appliquer en un autre point⁵. Lorsqu'une ou des théorie(s) de domaine permet(tent) de redécrire les situations, la notion de similarité doit faire intervenir des propriétés des redescription possibles et des théories qu'elles mettent en jeu. On parle alors vraiment d'analogies. Celles-ci présentent certaines particularités notables.

- Si l'analogie entre deux situations fait souvent intervenir la recherche de « points communs » entre elles, elle n'est cependant pas équivalente à un processus de généralisation. On peut plutôt l'envisager comme la recherche du canal de transmission le plus économique entre une description de situation et une autre. Cette recherche met en avant des primitives ou des structures de description potentiellement intéressantes parmi toutes les primitives et structures envisageables. En ceci elle aide déjà à la solution d'un problème donné par la focalisation sur les descriptions pertinentes.
- Contrairement à une distance, la notion de proximité mise en jeu dans l'analogie n'est pas symétrique, et corollairement ce qui est transporté d'une situation à l'autre n'est pas forcément la même chose dans un sens et dans l'autre.

Les modèles de raisonnement par analogie ont surtout été étudiés dans le cadre de recherches sur la cognition humaine. D'autres types de raisonnements permettant le transfert de connaissances entre domaines ont été également abordés, par exemple le *blending* [FT98] ou l'*effet tunnel cognitif* [CTC00]. Il reste encore beaucoup à faire pour comprendre pleinement ces types de raisonnements mêlant divers types de représentations, de connaissances et de d'inférences. Nous recommandons à ce sujet la lecture des ouvrages de Hofstadter, dont [Hof95].

5. Les mathématiciens seront tentés d'y reconnaître la notion de covariance sur des variétés différentielles.

6.5 Bilan

Cet ouvrage fait une place réduite aux apprentissages mettant en œuvre beaucoup de connaissances ou des théories du domaine. Ce n'est pas parce que les auteurs les considèrent comme négligeables. Au contraire, notre conviction est que l'apprentissage artificiel ne pourra faire l'économie de l'étude des apprentissages utilisant des connaissances. Cependant, pour des raisons à la fois historiques (en particulier l'irruption du connexionnisme dans les années quatre-vingt), scientifiques (il est tellement plus facile de faire des théorèmes dans des espaces numériques), institutionnelles (le *quantum* d'action en science est souvent celui d'un travail de thèse, or les systèmes à base de connaissances demandent des développements à beaucoup plus long terme) et économiques (il y a déjà fort à faire avec l'exploration préliminaire de bases de connaissances « nues » telles que les bases de données génomiques ou les bases de données des entreprises), l'étude des méthodes telles que l'apprentissage à partir d'explications, l'abstraction, l'analogie, etc. a fortement marqué le pas ces dernières années. Il ne fait cependant nul doute que dans dix ans un ouvrage sur l'apprentissage y consacrera une bien plus large place.

Résumé

Dès que l'on aborde des domaines complexes, il faut faciliter les raisonnements et l'apprentissage :

- en contrôlant l'expression de l'espace des hypothèses, par exemple en réalisant des abstractions,
- en apprenant des connaissances permettant une exploration plus efficace des hypothèses, par exemple à l'aide de macro-opérateurs ou d'heuristiques de contrôle qui rend possible l'apprentissage à partir d'explications,
- en facilitant le transfert de connaissances et de solutions entre domaines, par exemple en utilisant des raisonnements comme l'analogie.

Toutes ces techniques requièrent une ou des théories du domaine fortes. C'est de là qu'elles tirent une grande puissance en permettant l'apprentissage à partir de peu de données. C'est là aussi la source des difficultés de leur application.

Chapitre 7

L'inférence grammaticale

Ce chapitre traite de l'apprentissage inductif de grammaires et d'automates à partir d'exemples. Le problème est d'extraire, à partir d'exemples (et éventuellement de contre-exemples), une grammaire capable d'engendrer un langage, c'est-à-dire un ensemble de séquences, en général infini.

On se place ici dans le cas d'un apprentissage de concept : l'ensemble d'exemples est donc formé de deux parties, un échantillon positif, sous-ensemble fini du langage de la grammaire que l'on cherche à apprendre, et un échantillon négatif (éventuellement vide), ensemble fini de séquences n'appartenant pas au langage.

La décision de l'appartenance d'une phrase inconnue à une grammaire est réalisée par un algorithme d'analyse syntaxique, dont le rôle est de rendre une réponse binaire à la question : « telle séquence de symboles appartient-elle au langage engendré par telle grammaire ? »

L'inférence grammaticale a été étudiée depuis le développement (simultané) de la théorie des grammaires formelles et de l'intelligence artificielle. Cette discipline offre un grand intérêt théorique aux chercheurs en apprentissage. De plus, elle est l'objet de nombreuses applications. Nous consacrerons principalement ce chapitre à l'inférence régulière, c'est-à-dire à l'apprentissage des automates finis. C'est en effet dans ce cadre qu'ont été menés la plupart des travaux jusqu'à ce jour. Nous exposerons ensuite quelques extensions, en particulier aux grammaires algébriques.

LE ZOO vient d'acquérir deux cygnes, un noir et un blanc. Et chaque jour, le gardien transporte de sa cabane à leur cage six rations de farines animales, trois pour chacun. Comme il a déjà à s'occuper des ornithorynques et des rhinolophes, il décide de se simplifier la tâche : plutôt que de transporter les rations, il va essayer d'apprendre aux cygnes à venir se servir eux-mêmes. Qui plus est, cela fera une attraction dans l'établissement : à une heure dite, la porte s'ouvrira et les deux animaux quitteront leur cage pour se diriger vers leur repas. Avant de proposer le numéro, il faut dresser les oiseaux. Le gardien, qui a de bonnes lectures, décide de les conditionner de la manière suivante. Un jour sur quatre, il laisse l'un des oiseaux dans sa cage sans rien lui donner et emmène l'autre à sa cabane, où ce dernier a la bonne surprise de trouver non pas trois, mais cinq rations de pâtée. Un jour sur quatre, il fait la même chose en échangeant les oiseaux. Un jour sur quatre, il emmène les deux qui ont alors droit à deux rations et demi chacun. Le reste du temps, il leur apporte trois rations chacun dans leur cage, comme d'habitude. Quelques semaines de conditionnement, seul ou à deux, et l'expérience est prête. La porte s'ouvre. Que vont faire les cygnes ? Le tableau suivant montre le résumé de la situation : le cygne noir est noté *CN* et le blanc *CB*.

<i>CN</i> et <i>CB</i> se déplacent tous les deux	<i>CN</i> et <i>CB</i> ont chacun 2.5 rations
<i>CN</i> se déplace et <i>CB</i> ne se déplace pas	<i>CN</i> a 5 rations et <i>CB</i> n'a rien
<i>CN</i> ne se déplace pas et <i>CB</i> se déplace	<i>CN</i> n'a rien et <i>CB</i> a 5 rations
<i>CN</i> et <i>CB</i> ne se déplacent ni l'un ni l'autre	<i>CN</i> et <i>CB</i> ont chacun 3 rations

Chacun des cygnes a maintenant appris que s'il reste sur place, il aura peut-être trois rations, peut-être aucune, selon ce que fera l'autre. Il sait aussi que s'il fait seul l'effort de se déplacer, cinq rations seront peut-être pour lui. Cependant, si son camarade fait aussi le voyage, il n'y en aura que deux et demi pour chacun. La difficulté vient de ce qu'aucun des deux cygnes ne sait ce que va faire l'autre. Faut-il attendre et espérer que l'autre animal a fait de même pour recevoir les trois rations ? Faut-il essayer d'en avoir cinq, au risque que l'autre cherche aussi à manger plus, ce qui pénalise les deux ?

Le gardien a de bonnes raisons d'être optimiste. Chaque jour, pense-t-il, chaque cygne va raisonner comme suit : « je dois me déplacer, pour deux raisons. La première est que cela m'assure de manger au moins deux rations et demi. La seconde est que l'autre cygne va probablement penser comme moi et se déplacer aussi : il ne faut donc pas que je reste dans ma cage ». C'est en effet ce que l'on observe la plupart du temps : confrontés à des situations de ce type¹, deux sujets choisissent la sécurité. Le paradoxe apparent de la situation vient de ce que s'ils restaient tous deux dans leur cage, chacun aurait plus à manger. Mais comme ils n'ont pas la possibilité de se mettre d'accord, le résultat n'est pas finalement le meilleur pour eux.

Maintenant, nous allons voir comment des cygnes doués de capacités à vrai dire rares dans cette espèce peuvent mieux profiter de la situation. Nous allons faire deux hypothèses. La première est que *CB* possède une stratégie qui permet chaque jour de prendre une décision en fonction de ce qui s'est passé les jours précédents. La seconde est que *CN* connaît la famille de la stratégie que *CB* applique, mais qu'il ne sait pas exactement laquelle. Il va donc l'apprendre au fil des jours. Une fois qu'il l'aura induite du comportement de son adversaire, il lui sera facile de la déjouer et de manger plus. C'est le gardien qui va être surpris.

Commençons par *CB* pour présenter dans quelle famille de stratégies il se place. Le mieux est de représenter cela graphiquement. La figure 7.1 présente l'exemple de la stratégie naturelle qui consiste à se déplacer (sortir de la cage) chaque jour quelle que soit l'action effectuée la

1. Ce problème est souvent présenté par une autre métaphore : *le dilemme des prisonniers* ([Axe97, Del92, Gir00]).

veille par *CN*. C'est sur cette analyse que compte le gardien. Chaque jour, la décision consiste pour *CB* à se souvenir de ce que *CN* a fait la veille et à agir en conséquence. Pour cela, *CB*, qui s'est installé la veille dans un certain « état » (un cercle du diagramme) emprunte au jour *j* l'arc qui porte l'action faite la veille par son adversaire et se retrouve dans un nouvel « état ». Comme dans cette stratégie *CB* sort tous les jours, le diagramme n'a qu'un état. Chaque jour, *CB* s'y trouve, et quelle que soit l'action de *CN* la veille, *CB* va emprunter un des deux arcs pour prendre sa décision : évidemment celle de sortir. Mais le cygne blanc peut aussi vraiment

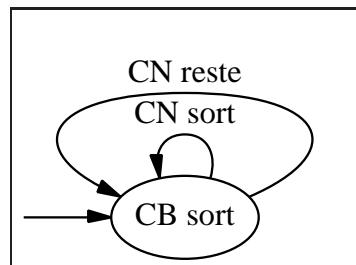


FIG. 7.1 – La stratégie triviale pour le cygne blanc *CB* : « je sors tous les jours, quelle qu'ait été l'action de mon adversaire la veille ».

tenir compte de ce que son partenaire *CN* a fait les jours précédents. En réalité, c'est la seule information observable². Par exemple, il peut décider de rester tous les jours dans sa cage, sauf si la veille *CN* est sorti. L'hypothèse de *CB* est ici est que si *CN* est sorti un jour, il va rester tranquille le lendemain. Cette stratégie est décrite par la figure 7.2 et le tableau ci-dessous.

Ce qui s'est passé au jour <i>j</i> – 1	La décision de <i>CB</i> au jour <i>j</i>
<i>CB</i> est resté et <i>CN</i> est sorti	<i>CB</i> sort
<i>CB</i> est resté et <i>CN</i> est resté	<i>CB</i> reste
<i>CB</i> est sorti et <i>CN</i> est sorti	<i>CB</i> reste
<i>CB</i> est sorti et <i>CN</i> est resté	<i>CB</i> reste

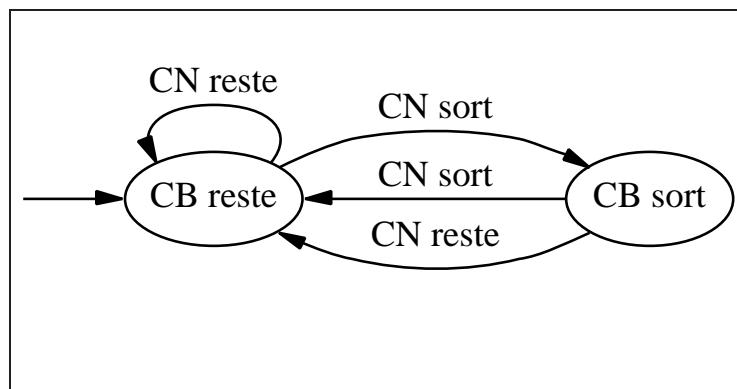


FIG. 7.2 – Une stratégie un peu plus complexe pour le cygne blanc *CB* : « Je reste tout les jours dans ma cage, sauf si mon adversaire est sorti hier. Dans ce cas, je pense qu'il va rester aujourd'hui et donc je sors ».

2. Il s'agit ici du dilemme itéré du prisonnier.

D'une manière générale, nous supposons donc que la stratégie du cygne blanc s'écrit sous la forme graphique dont nous avons déjà donné deux exemples. Ce genre de modèle s'appelle un automate fini. Chaque automate fini, donc chaque stratégie, correspond à la description pour le cygne blanc du concept « aujourd'hui, je sors » et de sa négation « aujourd'hui je reste », en fonction des couples d'actions commises par les deux animaux dans les jours précédents.

Supposons que la stratégie utilisée par *CB* soit celle de la figure 7.3. Comment *CB* utilise-t-il ce dessin pour agir en fonction de ce qu'a fait l'adversaire les jours précédents ? Appelons « états » les trois noeuds du graphe : l'un est en forme de cercle, l'autre de carré et le dernier de losange. Dans chacun est inscrite la décision à prendre. Le premier jour, le cygne blanc place sa stratégie dans l'état cercle. Il va donc sortir vers la cabane du gardien et observer ce que fait le cygne noir. Si celui-ci se déplace, la stratégie du cygne blanc reste dans le même état : le lendemain, il se déplacera encore. Par contre, si son adversaire reste dans la cage, la stratégie passera dans l'état rectangle, ce qui impose que le lendemain, le cygne blanc restera dans sa cage.

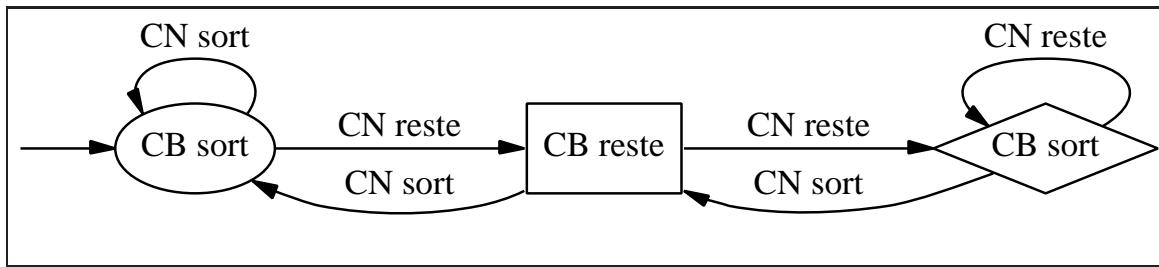


FIG. 7.3 – *La stratégie du cygne blanc.*

jour	action de <i>CB</i>	action de <i>CN</i>
1	sortir	sortir
2	sortir	rester
3	rester	sortir
4	sortir	sortir
5	sortir	rester
6	rester	rester
7	rester	rester
8	rester	sortir
9	rester	

TAB. 7.1 – *La séquence des actions journalières de *CB* selon sa stratégie, qui ne dépend que de ce que *CN* a fait les jours précédents.*

Sur une durée un peu plus longue, la séquence donnée dans le tableau ci-dessus est conforme à la stratégie du cygne blanc. En effet, le neuvième jour, la séquence des actions effectuée par le cygne noir les huit jours précédents conduit le cygne blanc (en suivant les flèches marquées dans le graphe) dans l'état en forme de rectangle, qui est marqué « *CB reste* ». D'où la décision prise par *CB* : rester. Le lendemain, neuvième jour, quelle que soit l'action du cygne noir le huitième, le cygne blanc restera dans sa cage : il sera soit dans l'état en forme de cercle, soit dans celui en forme de losange, mais la décision sera la même.

Nous avons admis que le cygne blanc possédait une stratégie dans la famille de celles que l'on peut représenter par un automate fini, comme les trois présentées ci-dessus. On a vu que cela

permet d'utiliser les actions passées de l'adversaire. Plaçons-nous maintenant du point de vue du cygne noir. Il n'a pas de stratégie *a priori*, mais il possède des facultés d'apprentissage qui vont lui permettre de déjouer celle de l'adversaire. Et comment? D'abord, il va apprendre quelle est la stratégie du cygne blanc, quel automate il utilise, en observant ses réactions quotidiennes. Il n'aura plus ensuite qu'à prendre la meilleure tactique, déduite de celle de l'adversaire.

Le premier point est le sujet de ce chapitre, l'inférence grammaticale: est-il possible d'apprendre un concept représenté sous la forme d'un automate fini quand on connaît des exemples de séquences conformes au concept et d'autres non couvertes par ce concept? La réponse est globalement positive, comme on le verra. Dans ce cas, l'ensemble d'apprentissage est constitué des huit exemples et contre-exemples donnés ci-dessous dans le tableau 7.2.

Séquence des actions de <i>CN</i>	Action de <i>CB</i>
(premier jour)	sortir
s	sortir
s r s	sortir
s r s s	sortir
s r	rester
s r s s r	rester
s r s s r r	rester
s r s s r r r	rester
s r s s r r r r	rester

TAB. 7.2 – *L'ensemble des séquences d'apprentissage qui permet à CN d'apprendre la stratégie de CB. On a abrégé « CN sort » en s et « CN reste » en r.*

Le second point provient de la théorie des jeux (le cygne noir est particulièrement savant). Il se trouve que, si on a la connaissance d'une stratégie adverse sous forme d'automate fini, on peut en déduire la meilleure contre-stratégie. Sans détailler le cas général, la meilleure réponse du cygne noir à la stratégie du cygne blanc est ici la suivante: rester dans la cage deux fois, puis alterner: rester, sortir, rester, sortir... un jour sur deux. Un calcul simple montre que la quantité de rations que le cygne noir avale par jour tend rapidement vers 4, la moyenne entre 3 et 5. L'apprentissage de la stratégie de l'adversaire par inférence grammaticale lui a donc été profitable, puisque s'il avait quitté sa cage chaque jour, la stratégie de *CB* lui aurait dicté de faire de même et *CN* n'aurait eu que deux rations et demi par jour.

Le plan de ce chapitre

Nous présentons à la section 7.1 les notions de base qui nous permettent de définir l'espace des solutions possibles de l'inférence grammaticale régulière. Nous introduisons aussi la notion de *complétude structurelle* d'un échantillon relativement à un automate fini, un biais d'apprentissage qui assure d'une certaine manière que cet échantillon est représentatif de l'automate.

L'inférence grammaticale est l'un des domaines de l'apprentissage automatique symbolique pour lequel la caractérisation et l'évaluation d'une méthode ont été le plus formalisées. Nous présentons en particulier le critère d'*identification exacte* d'un langage, défini par Gold [Gol67b, Gol78b].

Nous consacrons la section 7.3 à une description formelle de l'inférence régulière. Elle est vue comme un problème de *recherche explicite* dans l'espace des hypothèses de généralisation de l'échantillon positif. Compte tenu d'une limitation raisonnable des possibilités de généralisation, l'espace de recherche à explorer est un ensemble d'automates structuré construit sur l'échantillon positif.

De nombreux algorithmes d'inférence ont été proposés. Certains sont *constructifs* ou *caractérisables* au sens où ils sont capables d'identifier *avec certitude* n'importe quel langage appartenant à une *classe définie* (un sous-ensemble strict de langages réguliers défini de manière constructive). D'autres d'algorithmes sont qualifiés ici d'*heuristiques*, soit parce qu'ils n'identifient pas avec certitude n'importe quel langage d'une classe définie, soit parce qu'il n'existe pas de caractérisation de la classe de langages qu'ils peuvent identifier. Ces algorithmes utilisent un *biais d'apprentissage* heuristique pour guider la généralisation effectuée à partir des données.

Certains algorithmes d'inférence régulière n'utilisent que des échantillons positifs. La plupart de ces algorithmes sont décrits dans les articles de synthèse [BF72a, Ang82a, Mic90, Gre94a]. Nous présentons à la section 7.4 quelques algorithmes constructifs ou heuristiques qui utilisent ce type de données.

Si on dispose également d'un échantillon négatif, l'inférence peut être plus précise et on peut en particulier chercher à construire *le plus petit automate compatible (cohérent)* avec les échantillons positif et négatif. Nous étudions le cadre formel de ce problème à la section 7.5. Nous définirons pour cela la notion d'*ensemble frontière* (section 7.5.1). Nous présenterons ensuite quelques méthodes d'inférence.

La section suivante traitera plus brièvement du problème de l'inférence de grammaires algébriques, un problème encore mal maîtrisé. On y verra quelques exemples d'algorithmes heuristiques. La dernière section abordera certaines extensions et problèmes ouverts de ce domaine, qui est à l'heure actuelle une véritable mine théorique et pratique pour les chercheurs.

Notations utiles pour le chapitre

Σ	Un alphabet (ensemble fini)
ϵ	Le mot vide
Σ^*	L'ensemble de tous les mots sur Σ
Σ^+	Σ^* sauf le mot vide ϵ
Σ^n	Tous les mots sur Σ de longueur inférieure ou égale à n

7.1 Définitions et notations

7.1.1 Langages, grammaires, automates et partitions

Notions de base

Définition 7.1

Soit Σ un ensemble fini, appelé *alphabet* et les lettres a, b, c, \dots les éléments de Σ . On note u, v, w, x des éléments de Σ^* , c'est-à-dire des chaînes ou phrases³ de longueur dénombrable sur Σ . On note ϵ la chaîne vide de longueur nulle et $|u|$ la longueur de la chaîne u .

Définition 7.2

u est un préfixe de v s'il existe w tel que $uw = v$.

Définition 7.3

Un langage L est un sous-ensemble quelconque de Σ^* . Les éléments de L sont des séquences de lettres de Σ , donc des chaînes.

3. On dit aussi *mots* ou *séquences*.

Grammaires

Une *grammaire* est un objet mathématique auquel est associé un processus algorithmique permettant d'engendrer un langage.

Définition 7.4

Une grammaire est un quadruplet $G = (N, \Sigma, P, S)$ dans lequel :

- N est un alphabet composant l'ensemble des symboles non-terminaux de G .
- Σ est l'alphabet terminal de G , disjoint de N . On note : $V = N \cup \Sigma$.
- $P \subseteq (V^* N^+ V^* \times V^*)$ est un ensemble fini de règles de production.
- $S \in N$ est l'axiome de G .

Une règle de production s'écrit $\alpha \rightarrow \beta$, avec $\beta \in V^*$ et $\alpha \in V^* N^+ V^*$, ce qui signifie que α comporte au moins un symbole non-terminal.

Génération d'un langage par une grammaire

Soit $u \in V^+$ et⁴ $v \in V^*$. On dit que u se réécrit en v selon la grammaire G , ou que la grammaire G dérive v de u en une étape si et seulement si on peut écrire u et v sous la forme :

- $u = xu'y$ (avec éventuellement $x = \epsilon$ ou $y = \epsilon$)
- $v = xv'y$
- avec : $(u' \rightarrow v') \in P$

La grammaire G dérive v en k étapes de u si et seulement s'il existe $k \geq 1$ et une suite $(v_0 \dots v_k)$ de mots de V^+ tels que :

- $u = v_0$
- $v = v_k$
- v_i se récrit en v_{i+1} pour $0 \leq i \leq k - 1$

La grammaire G dérive v de u s'il existe un entier k tel que u dérive v en k étapes. Pour $k \geq 1$, la séquence u, \dots, v_i, \dots, v s'appelle une *dérivation* de v par u .

Définition 7.5

On dit qu'un mot $v \in \Sigma^*$ est engendré par la grammaire G quand il peut se dériver à partir de l'axiome S de G .

Le langage engendré par la grammaire G est l'ensemble des mots de Σ^* engendrés par G . On le note $L(G)$.

Deux exemples

Quand il n'y a pas d'ambiguïté, on simplifie la notation d'une grammaire en ne représentant que ses règles et en mettant toutes celles de même partie gauche sur la même ligne.

La grammaire définie par $N = \{S\}$, $\Sigma = \{a, b, c\}$ et $P = \{(S \rightarrow aSb), (S \rightarrow \epsilon)\}$ peut ainsi s'écrire :

$$S \rightarrow aSb \mid \epsilon$$

Elle engendre le langage $\{a^n b^n \mid n \geq 0\}$. En effet, sur un exemple, son axiome S dérive le mot $aaabbb$ en quatre étapes : trois applications de la règle $S \rightarrow aSb$ et une de la règle $S \rightarrow \epsilon$, soit :

$$S, aSb, aaSbb, aaaSbbb, aaabbb$$

4. Étant donné un alphabet V , V^* désigne l'ensemble de tous les mots sur V , alors que V^+ exclut le mot vide.

La grammaire définie par $N = \{1, 2, 3, 4\}$, $\Sigma = \{a, b\}$ et $P = \{(1 \rightarrow a2), (1 \rightarrow b3), (1 \rightarrow \epsilon), (2 \rightarrow a1), (2 \rightarrow b4), (3 \rightarrow a4), (3 \rightarrow b1), (4 \rightarrow a3), (4 \rightarrow b2)\}$ est exactement équivalente à l'automate fini de la figure 7.4. Les phrases qu'elle accepte sont composées d'un nombre pair de a et pair de b . Elle s'écrit plus simplement⁵ :

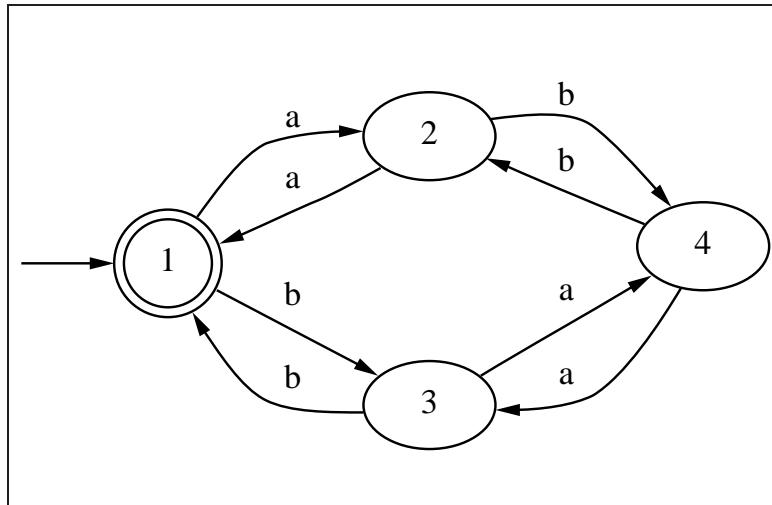


FIG. 7.4 – Un automate fini acceptant les phrases composées d'un nombre pair de a et pair de b .

$$\begin{array}{ll} 1 \rightarrow a2 \mid b3 \mid \epsilon & 2 \rightarrow a1 \mid b4 \\ 3 \rightarrow a4 \mid b1 & 4 \rightarrow a3 \mid b2 \end{array}$$

Deux types de grammaires

Selon la forme de leur règles $\alpha \rightarrow \beta$, on distingue les grammaires suivantes :

Type 2 : grammaires algébriques⁶ : $A \rightarrow \beta$, avec $A \in N$ et $\beta \in V^*$

Type 3 : grammaires régulières : $A \rightarrow wB$ ou $A \rightarrow w$, avec $w \in \Sigma^*$, $A \in N$ et $B \in N$

Un langage pouvant être engendré par une grammaire régulière (resp : algébrique) est appelé *langage régulier* (resp : *langage algébrique*). Un résultat classique de la théorie des langages est le suivant ([AU72]):

Théorème 7.1

Tout langage régulier peut être engendré par un automate fini. Tout automate fini engendre un langage régulier.

Les automates finis

Comme l'assure le théorème précédent, les automates finis sont équivalents aux grammaires régulières. Ils sont d'un emploi beaucoup plus facile. Nous allons maintenant les définir rigoureusement.

Définition 7.6

Un automate fini est un quintuplet $(Q, \Sigma, \delta, q_0, F)$ où Q est un ensemble fini d'états, Σ est un alphabet fini, δ est une fonction de transition, c'est-à-dire une application de $Q \times \Sigma \rightarrow 2^Q$,

5. L'axiome n'est pas noté ici S , mais 1.

6. Les grammaires algébriques sont aussi appelées *context-free* ou *hors-contexte*.

$Q_0 \in Q$ est le sous-ensemble des états initiaux et $F \in Q$ est le sous-ensemble des états finaux ou d'acceptation.

Définition 7.7

Si, pour tout q de Q et tout a de Σ , $\delta(q, a)$ contient au plus un élément (respectivement exactement un élément) et si Q_0 ne possède qu'un élément q_0 , l'automate A est dit déterministe (respectivement complet). Par la suite, nous utiliserons l'abréviation AFD pour « automate fini déterministe » et AFN pour un « automate fini non-déterministe ».

L'automate fini de la figure 7.4 est un AFD. Un autre exemple d'automate fini est représenté à la figure 7.5. Il comporte cinq états, $Q = \{0, 1, 2, 3, 4\}$. Il est défini sur l'alphabet à deux lettres, $\Sigma = \{a, b\}$. Les états initiaux sont 0 et 5, $Q_0 = \{0, 5\}$ et les états 3 et 4 sont finaux, $F = \{3, 4\}$.

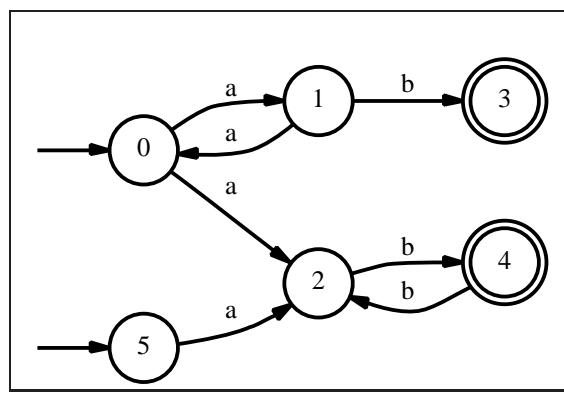


FIG. 7.5 – Une représentation de l'automate A_1 .

Il s'agit ici d'un AFN (automate non déterministe) car il y a deux arcs étiquetés a à partir de l'état 0 et deux états initiaux. En d'autres termes, $\delta(0, a) = \{1, 2\}$.

Langage accepté par un automate fini

Définition 7.8

Une acceptation d'une chaîne $u = a_1 \dots a_l$ par un automate A , éventuellement non-déterministe, définit une séquence, éventuellement non unique, de $l + 1$ états (q^0, \dots, q^l) telle que $q^0 \in Q_0$, $q^l \in F$ et $q^{i+1} \in \delta(q^i, a_{i+1})$, pour $0 \leq i \leq l - 1$. Les $l + 1$ états sont dits atteints pour cette acceptation et l'état q^l est utilisé comme état d'acceptation. De façon similaire, les l transitions, c'est-à-dire des éléments de δ , sont dites exercées par cette acceptation.

Par exemple, la séquence des états $(0, 1, 3)$ correspond à une acceptation de la chaîne aab dans l'automate A_1 .

Définition 7.9

Le langage $L(A)$ accepté par un automate A est l'ensemble des chaînes acceptées par A .

Théorème 7.2

Le langage L est accepté par un automate A si et seulement si il peut être engendré par une grammaire régulière. De plus, un langage régulier peut être représenté par une expression régulière⁷

7. Nous ne définissons pas formellement ici ce terme.

[AU72].

Définition 7.10

Pour tout langage régulier L , il existe un AFD noté $A(L)$ qui engendre L et possède un nombre minimal d'états. $A(L)$ est généralement appelé automate déterministe minimal ou automate canonique de L . On démontre que $A(L)$ est unique [AU72]. Par la suite, nous parlerons indifféremment de l'automate canonique ou de l'automate minimal déterministe de L .

Par exemple, l'automate canonique représenté à la figure 7.6 accepte le langage composé des phrases commençant par un nombre impair de a , suivi d'un nombre impair de b ⁸. Il s'agit du langage accepté également par l'automate A_1 de la figure 7.5. Il n'existe pas d'automate déterministe comportant moins d'états et acceptant ce langage.

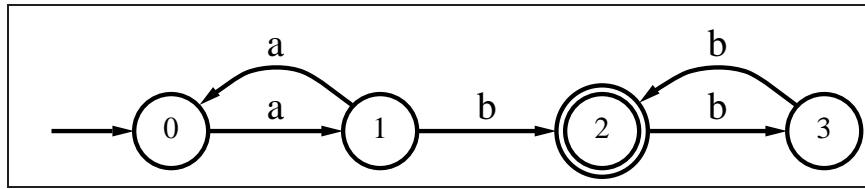


FIG. 7.6 – Automate canonique du langage défini par l'expression régulière $L = a(aa)^*b(bb)^*$.

Automates dérivés

Nous définissons maintenant une relation d'ordre partiel sur l'ensemble des automates, qui permettra un apprentissage par généralisation dans l'esprit de la méthode de l'espace des versions.

Définition 7.11

Pour tout ensemble S , une partition π est un ensemble de sous-ensembles de S , non vides et disjoints deux à deux, dont l'union est S . Si s désigne un élément de S , $B(s, \pi)$ désigne l'unique élément, ou bloc, de π comprenant s . Une partition π_i raffine, ou est plus fine que, une partition π_j ssi tout bloc de π_j est un bloc de π_i ou est l'union de plusieurs blocs de π_i .

Définition 7.12

Si $A = (Q, \Sigma, \delta, q_0, F)$ est un automate,

l'automate $A/\pi = (Q', \Sigma, \delta', B(q_0, \pi), F')$ dérivé de A relativement à la partition π de Q , aussi appelé l'automate quotient A/π , est défini comme suit :

$$Q' = Q/\pi = \{B(q, \pi) | q \in Q\},$$

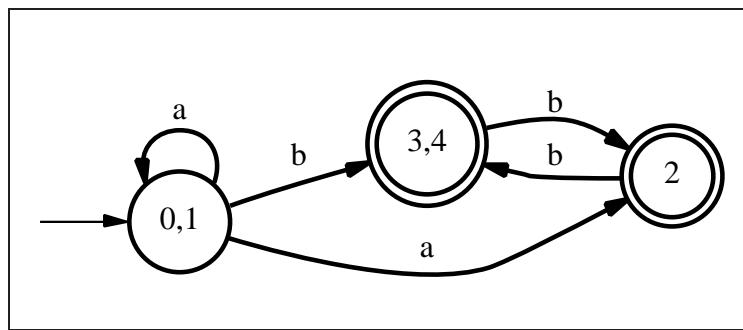
$$F' = \{B \in Q' | B \cap F \neq \emptyset\},$$

$$\delta' : Q' \times \Sigma \rightarrow 2^{Q'} : \forall B, B' \in Q', \forall a \in \Sigma, B' \in \delta'(B, a) \text{ssi } \exists q, q' \in Q, q \in B, q' \in B' \text{ et } q' \in \delta(q, a)$$

Nous dirons que les états de Q appartenant au même bloc B de la partition π sont fusionnés.

Reprendons l'automate A_1 , représenté à la figure 7.5 et définissons la partition de son ensemble d'états, $\pi_2 = \{\{0, 1\}, \{2\}, \{3, 4\}\}$. L'automate quotient A_1/π_2 , obtenu en fusionnant tous les états appartenant à un même bloc de π_2 , est représenté à la figure 7.7.

8. ce qui correspond à l'expression régulière $L = a(aa)^*b(bb)^*$.

FIG. 7.7 – L'automate quotient A_1/π_2 .

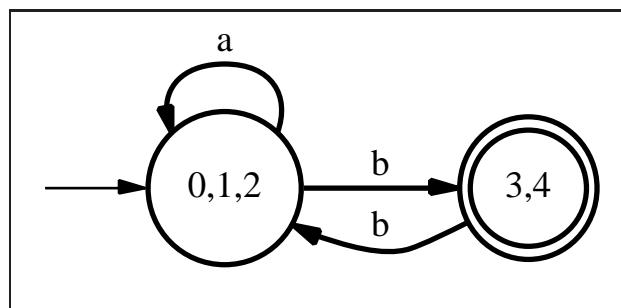
Une propriété fondamentale de l'opération de fusion est la suivante:

Propriété 7.1

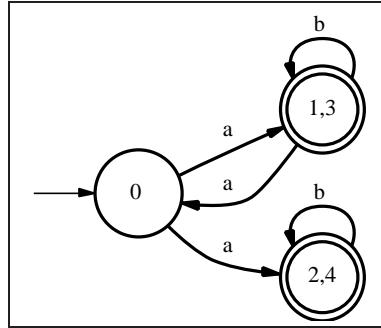
Si un automate A/π_j dérive d'un automate A/π_i , alors le langage accepté par A/π_i est inclus dans celui accepté par A/π_j .

Par conséquent, en partant d'un automate A , on peut construire tous les automates dérivés de A en énumérant les partitions des états de A . Il existe sur cet ensemble une relation d'ordre partiel qui est cohérente avec l'inclusion des langages que reconnaissent ces automates. On reconnaît ici la relation de généralité de l'espace des versions (voir le chapitre 4).

En reprenant l'exemple A_1 de la figure 7.5, on a vu que le choix de la partition $\pi_2 = \{\{0, 1\}, \{2\}, \{3, 4\}\}$ permet de dériver l'automate quotient $A_2 = A_1/\pi_2$, représenté à la figure 7.7. On sait donc que le langage reconnu par A_2 inclut celui reconnu par A_1 . Prenons maintenant la partition $\pi_3 = \{\{0, 1, 2\}, \{3, 4\}\}$. Elle permet de dériver un automate A_3 , représenté à la figure 7.8.

FIG. 7.8 – L'automate quotient A_1/π_3 .

La partition π_3 est *moins fine* que π_2 , puisque ses blocs sont construits comme une union de blocs de π_2 ; on peut donc assurer que l'automate A_3 reconnaît un langage qui inclut celui reconnu par A_2 . En revanche, si on construit l'automate A_4 (figure 7.9) par dérivation de A_1 selon la partition $\pi_4 = \{\{0\}, \{1, 3\}, \{2, 4\}\}$, qui n'est ni moins fine ni plus fine que π_2 , on ne peut rien dire sur l'inclusion éventuelle des langages reconnus par A_4 et A_2 . Par exemple, la phrase abb est reconnue par A_4 et pas par A_2 , alors que la phrase b est reconnue par A_2 et pas par A_4 .

FIG. 7.9 – L’automate quotient A_1/π_4 .**Propriété 7.2**

L’ensemble des automates dérivés d’un automate A , qui est partiellement ordonné par la relation de dérivation, est un treillis⁹. Les automates A et UA , l’automate universel, en sont respectivement les éléments minimal et maximal. On note ce treillis $\text{Lat}(A)$.

On retrouve donc ici la structure en treillis de l’espace des hypothèses, associée à une relation d’ordre de généralité, comme définie au chapitre 4.

7.1.2 Échantillons d’un langage et automates associés

Définition 7.13

Nous désignons¹⁰ par I_+ un sous-ensemble fini, appelé échantillon positif, d’un langage L quelconque. Nous désignons par I_- un sous-ensemble fini, appelé échantillon négatif, du langage complémentaire $\Sigma^* - L$. I_+ et I_- sont des sous-ensembles finis et disjoints de Σ^* .

Complétude structurelle

On définit maintenant un biais d’apprentissage : on ne cherchera à inférer que des automates pour lesquels l’échantillon positif est d’une certaine manière représentatif (en termes techniques, *structurellement complet*). Ce biais est conforme à un principe d’économie de type « rasoir d’Occam » ou principe *MDL* : il est inutile de rechercher des automates trop complexes.

Définition 7.14

Un échantillon I_+ est structurellement complet relativement à un automate A acceptant L , s’il existe une acceptation $\mathcal{AC}(I_+, A)$ de I_+ telle que :

- Toute transition de A soit exercée.
- Tout élément de F (l’ensemble des états finaux de A) soit utilisé comme état d’acceptation.

Par exemple, l’échantillon $I_+ = \{aab, ab, abbbb\}$ est structurellement complet relativement à l’automate A_1 .

L’automate Canonique Maximal, l’Arbre Accepteur des Préfixes et l’Automate Universel

9. Cet ensemble n’est pas un treillis au sens algébrique, mais l’usage a jusqu’ici fait prévaloir ce terme.

10. En inférence grammaticale, les notations I_+ et I_- sont classiques. Nous les employons donc dans ce chapitre à la place de S_+ et S_- .

On a maintenant besoin de représenter l'échantillon d'apprentissage dans l'univers des concepts, c'est-à-dire de définir des automates qui ne reconnaissent que cet échantillon.

Définition 7.15

On désigne par $MCA(I_+) = (Q, \Sigma, \delta, q_0, F)$ l'automate maximal canonique¹¹ relatif à I_+ [Mic80a].

Par construction, $L(MCA(I_+)) = I_+$ et $MCA(I_+)$ est le plus grand automate (l'automate ayant le plus grand nombre d'états) pour lequel I_+ est structurellement complet. $MCA(I_+)$ est généralement non-déterministe.

Sa construction est facile à observer sur un exemple. L'automate représenté à la figure 7.10 est l'automate maximal canonique relatif à l'échantillon $I_+ = \{a, ab, bab\}$.

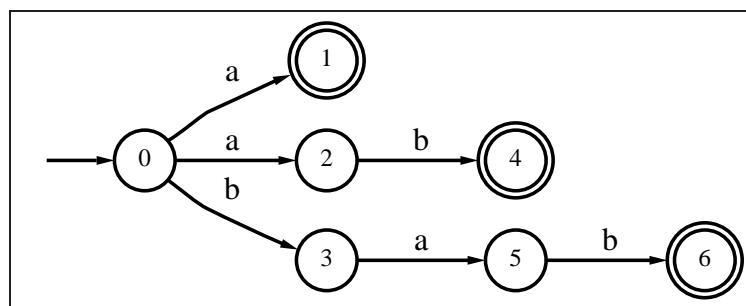


FIG. 7.10 – L'automate maximal canonique de l'échantillon $\{a, ab, bab\}$.

Définition 7.16

Nous désignons par $PTA(I_+)$ l'arbre accepteur des préfixes¹² de I_+ [Ang82a]. Il s'agit de l'automate quotient $MCA(I_+)/\pi_{I_+}$ où la partition π_{I_+} est définie comme suit :

$$B(q, \pi_{I_+}) = B(q', \pi_{I_+}) \text{ssi } Pr(q) = Pr(q').$$

En d'autres termes, $PTA(I_+)$ peut être obtenu à partir du $MCA(I_+)$ en fusionnant les états partageant les mêmes préfixes. $PTA(I_+)$ est déterministe.

À titre d'exemple, l'automate représenté à la figure 7.11 est l'arbre accepteur des préfixes relatif à l'échantillon $I_+ = \{a, ab, bab\}$.

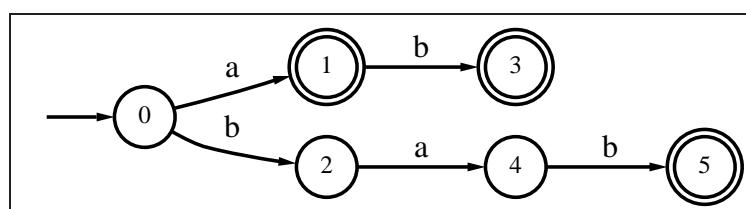


FIG. 7.11 – L'arbre accepteur des préfixes $PTA(I_+)$, avec $I_+ = \{a, ab, bab\}$.

11. L'abréviation **MCA** pour *Maximal Canonical Automaton* provient de la terminologie anglaise. Le qualificatif *canonique* se rapporte ici à un échantillon. Le MCA ne doit pas être confondu avec l'automate canonique d'un langage (voir déf. 7.10).

12. L'abréviation **PTA** pour *Prefix Tree Acceptor* provient de la terminologie anglaise.

Définition 7.17

Nous désignons par UA l'automate universel¹³. Il accepte toutes les chaînes définies sur l'alphabet Σ , c'est-à-dire $L(UA) = \Sigma^*$. Il s'agit donc du plus petit automate pour lequel tout échantillon de Σ^* est structurellement complet.

L'automate universel défini sur l'alphabet $\Sigma = \{a, b\}$ est représenté à la figure 7.12.

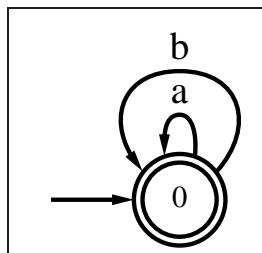


FIG. 7.12 – L'automate universel sur l'alphabet $\Sigma = \{a, b\}$.

7.2 Les protocoles de l'inférence : quelques résultats théoriques

7.2.1 La spécification d'un problème d'inférence grammaticale

Si l'on veut définir un cadre aussi général et productif que possible pour l'inférence grammaticale, comme d'ailleurs pour tout problème d'apprentissage inductif, cinq points doivent être spécifiés :

- La *classe de grammaires* à inférer : grammaires régulières¹⁴ ou d'une de ses sous-classes, ou grammaires hors-contextes. Comme on le verra en conclusion, quelques autres familles de grammaires ont aussi été étudiées.
- L'*espace des hypothèses*, ou le *langage des concepts*, c'est-à-dire l'ensemble des descriptions tel que chaque grammaire de la classe considérée possède au moins une description dans l'espace. Pour les grammaires régulières, il s'agit en général de l'espace des *automates*. Pour les grammaires hors-contexte, l'espace de représentation est en général directement celui de leurs règles.
- Un *ensemble d'exemples*, pour chaque grammaire à inférer et un *protocole* de présentation de ces exemples. Ce terme désigne la façon dont les exemples sont présentés à l'algorithme d'apprentissage : tous à la fois, un par un ...
- La *classe des méthodes d'inférence* en considération. Il s'agit en particulier du choix entre méthodes constructives ou heuristiques (voir section 7.4).
- Les *critères* d'une inférence réussie, définis en section 7.2.2.

Une *méthode d'inférence* est alors un processus calculable qui lit des exemples conformément à un protocole et propose des solutions dans l'espace des hypothèses.

13. L'abréviation UA pour *Universal Automaton* provient de la terminologie anglaise.

14. Rappelons qu'une grammaire régulière, équivalente à un automate fini, engendre un langage régulier (voir section 7.1).

7.2.2 L'identification à la limite d'une grammaire

Pour illustrer les notions ci-dessus et présenter un cadre théorique de référence, nous présentons dans ce paragraphe un protocole de présentation et un critère de réussite définis conjointement : l'*identification à la limite*, proposée par Gold [Gol67b]. De nombreux résultats découlent de ce critère, tant sur la possibilité d'identifier ou non un langage que sur la complexité calculatoire de ce processus d'identification [Gol78b, Pit89]. En outre, ces travaux constituent une justification théorique de la nécessité d'utiliser un échantillon négatif.

Définition 7.18 (présentation positive)

Une présentation positive d'un langage L est une séquence infinie de phrases de L comportant au moins une occurrence de chaque élément de L . Les éléments d'une présentation positive sont appelés des exemples.

Définition 7.19 (présentation négative)

Une présentation négative d'un langage L est une séquence infinie d'éléments de $\Sigma^ - L$ comportant au moins une occurrence de chaque élément de $\Sigma^* - L$. Les éléments d'une présentation négative sont appelés exemples négatifs ou contre-exemples.*

Définition 7.20 (présentation complète)

Une présentation complète d'un langage L est une séquence infinie de paires (x, d) , de $\Sigma^ \times \{0, 1\}$ telle que chaque x de L apparaît dans au moins une paire. Pour chaque (x, d) , $d = 1$ si et seulement si x appartient à L , $d = 0$ sinon.*

Le protocole d'identification à la limite est défini comme suit. Soit M une méthode d'inférence et $S = (x_1, d_1), (x_2, d_2) \dots$ une présentation complète d'exemples d'un langage L . Nous notons S_i le sous-ensemble fini des i premiers éléments de S selon un certain protocole. La méthode d'inférence doit proposer une solution dans l'espace d'hypothèses après chaque nouvel exemple lu. Soit $h(S_i)$, l'hypothèse proposée par la méthode M après avoir lu les i premiers exemples présentés.

Définition 7.21

La méthode d'inférence M identifie à la limite pour la classe de langages \mathcal{L} pour une classe de présentation \mathcal{P} si pour tout L de \mathcal{L} et tout p de \mathcal{P} , il existe un indice fini j , tel que :

- $h(S_i) = h(S_j)$, pour tout $i \geq j$.
- $L(h(S_j)) = L$.

La première condition impose que la méthode converge à partir d'un indice fini, la seconde qu'elle converge vers une représentation unique de la solution correcte.

Définition 7.22

Une méthode d'inférence M identifie à la limite une classe \mathcal{C} de langages, si M identifie à la limite n'importe quel langage L de la classe \mathcal{C} pour n'importe quelle présentation d'exemples de L .

Ce protocole s'intéresse donc à l'*identification exacte* d'un langage.

7.2.3 Deux propriétés de l'identification à la limite.

Propriété 7.3

La classe des langages réguliers¹⁵ n'est pas identifiable à la limite à partir d'une présentation positive.

Propriété 7.4

La classe des langages réguliers et celle des langages algébriques¹⁶ sont identifiables à la limite par présentation complète.

7.2.4 Autres protocoles pour l'inférence de grammaires.

La richesse théorique des l'apprentissage des grammaires formelles se traduit en particulier par la variété des protocoles proposés et les résultats démontrés dans ces protocoles. Citons par exemple la technique *par oracle*, dont une variante est la suivante : à chaque étape, l'algorithme d'inférence calcule une grammaire hypothèse et la propose à l'oracle. Celui-ci répond « oui » si l'hypothèse est la bonne, et sinon, donne un contre-exemple à l'hypothèse proposée, ce qui permet à l'algorithme de calculer une nouvelle hypothèse. L'algorithme peut aussi proposer une séquence et l'oracle lui répond si oui ou non elle appartient au langage à identifier. Il a été démontré que dans ce cadre, l'identification des automates déterministes est possible avec un nombre de questions à l'oracle polynomial en fonction de la taille de l'automate à trouver et de la taille du plus long contre-exemple.

On verra au paragraphe 7.6.4 une méthode qui apprend des grammaires algébriques à l'aide d'un oracle (aux réponses un peu différentes).

D'un point de vue pratique, on doit en général admettre que les protocoles de l'identification à la limite ou de l'intervention d'un oracle sont inutilisables. On se contente d'observer des exemples et de calculer une grammaire qui les généralise. La validité de l'inférence peut être bien sûr vérifiée par un ensemble de test.

Il existe par exemple un serveur Internet nommé **abbadingo** [LP97] qui fonctionne ainsi : l'auteur d'un algorithme d'inférence régulière demande à ce serveur une session de test. Quelques paramètres sont précisés : nombre d'états de l'automate canonique, taille de l'alphabet, « densité » de l'échantillon. L'automate est fabriqué aléatoirement par le serveur, ainsi que deux échantillons d'apprentissage et de test. Le premier est donné à l'utilisateur, qui fait tourner son algorithme et renvoie une grammaire en réponse. Le serveur l'utilise pour analyser l'échantillon de test et retourne la matrice de confusion (voir au Chapitre 3 le paragraphe 3.4). Il est à noter que la validité statistique de ce test n'est pas assurée ici, car on ne peut pas savoir si les séquences engendrées aléatoirement sont indépendantes.

On connaît un théorème (décevant) sur la recherche de l'automate fini « optimal », l'automate fini déterministe minimal $A(L)$, que nous avons aussi appelé *automate canonique* d'un langage L (7.10). Ce problème est connu sous le nom de *recherche du plus petit automate compatible*¹⁷.

Théorème 7.3

Étant donné un langage L , identifier $A(L)$ à partir d'exemples positifs et négatifs est un problème NP-difficile.

15. Le résultat de Gold est plus général, mais cette version suffira ici.

16. Ici encore, Gold a un résultat plus puissant.

17. *compatible* signifie simplement que toutes les phrases de I_+ sont acceptées (donc que l'automate est correct et complet (voir au chapitre 3) et toutes celles de I_- rejetées. C'est un synonyme de « cohérent ».

C'est-à-dire qu'il n'existe vraisemblablement pas d'algorithme qui puisse résoudre ce problème en un temps polynomial en fonction du nombre d'états de $A(L)$. Mais cela n'empêche pas de chercher des heuristiques qui permettent de trouver en un temps raisonnable un automate de bonne qualité (au sens du protocole **abbadingo**, par exemple).

7.2.5 L'inférence grammaticale et l'apprentissage *PAC*

Nous avons vu dans la section précédente les résultats négatifs concernant l'identification à la limite proposée par Gold. Nous allons présenter d'autres résultats dans le cadre *PAC* (voir le chapitre 2). Bien que très contraignant¹⁸, ce cadre est intéressant car il possède des versions affaiblies dans lesquelles l'approximation des langages réguliers est possible polynomialement par présentation complète.

7.2.5.1 Apprentissage *PAC*

Le cadre *PAC* original proposé par Valiant [Val84c] a subi de nombreuses adaptations et extensions depuis sa création [BEHW89, Hau88, KV89, Pit89, KV89]. Nous présentons ici la version proposée par Pitt [Pit89] qui est adaptée à l'inférence de langages réguliers (plus particulièrement, dans cette version, Pitt considère que les langages réguliers sont représentés par des automates finis déterministes (DFA)).

Nous rappelons ici une description informelle du modèle *PAC*, puis nous en donnons une version adaptée à l'inférence grammaticale. Rappelons que ce concept a été décrit au chapitre 2.

Soit L un langage défini sur Σ^* . Soit h un concept appris sur L . On dit que h est correct¹⁹ pour L si, pour tout mot u de Σ^* , h arrive à dire si u appartient ou non à L . On dira que h est approximativement correct si la probabilité qu'il classe mal un mot est « petite ».

Un algorithme d'apprentissage est correct (respectivement approximativement correct) si pour tout langage L défini sur Σ^* , le concept qu'il apprend est correct (respectivement approximativement correct) pour L .

On dit qu'un algorithme d'apprentissage est probablement approximativement correct si la probabilité qu'il apprenne un concept approximativement correct est grande.

Valiant a formalisé cette notion intuitive en créant le critère *PAC*: *Probablement Approximativement Correct*. Cette première version ne prend pas en compte la complexité temporelle et spatiale nécessaire à l'inférence. Lorsque le temps d'apprentissage n'est pas pris en compte, ce critère est trop permissif et considère que l'apprentissage d'algorithmes triviaux est satisfaisant. Nous allons donner maintenant la version formelle de l'apprentissage *PAC* polynomial (aussi appelée poly-*PAC*) comme l'a défini Pitt [Pit89].

Définition 7.23 (Identification *PAC* [Pit89])

- Soit M un DFA de taille²⁰ n .
- Soient ϵ et δ deux réels strictement positifs.
- Soit une distribution de probabilités D définie sur les chaînes de Σ^* de taille au plus p .

18. Ce cadre est contraignant dans le sens où il impose des résultats quelle que soit la distribution de probabilités selon laquelle les exemples sont distribués.

19. Dans les autres chapitres, en particulier 2 3 4, le mot « cohérent » a été employé. Nous gardons ici provisoirement « correct » pour faire la transition avec l'apprentissage *PAC*.

20. Le critère classiquement utilisé pour mesurer la taille d'un DFA est le nombre de ses états.

- Soit un algorithme (éventuellement aléatoire) A qui prend en entrée un ensemble de mots tirés conformément à la distribution D et étiquetés correctement par M .
- Soit l le nombre d'exemples obtenus par A .

A PAC-identifie les DFA si, quel que soit M , quel que soit D , quel que soit m , A produit un DFA M' en temps polynomial en $n, p, l, \frac{1}{\epsilon}, \frac{1}{\delta}, |\Sigma|$ tel que²¹ :

$$\Pr\left(\sum_{x \in L(M) \oplus L(M')} \Pr_D(x) < \epsilon\right) > 1 - \delta$$

On est ici dans le cas de l'apprentissage d'un concept sur un ensemble fini, puisque la taille des phrases est bornée par p . Un résultat supplémentaire sur les automates finis permet de considérer ce cas comme raisonnable dans un contexte d'approximation : pour tout $\gamma > 0$, il existe un entier p tel que la probabilité qu'une chaîne de longueur supérieure à p soit tirée est inférieure à γ .

7.2.5.2 Comparaison de l'apprentissage PAC et de l'identification à la limite

Quelques points de comparaison entre ces deux méthodes peuvent être soulignés :

- L'identification à la limite impose de retrouver exactement une représentation du langage cible. Le cadre PAC pour sa part se contente d'une approximation de la cible.
- L'identification à la limite ne pose aucune contrainte sur la complexité calculatoire des algorithmes alors que l'approximation PAC demande la polynomialité de l'inférence en fonction de la précision, de la tolérance imposée au résultat ainsi que de la taille de la cible et celle de Σ , et d'une borne sur la plus grande chaîne.
- L'identification à la limite ne garantit d'aucune manière la vitesse de convergence alors que l'approximation PAC impose que la précision de l'inférence soit dépendante de la quantité de données.
- Enfin, le critère de Valiant amène la notion de distribution de probabilités des exemples qui n'existe pas dans le critère de Gold. Il impose cependant que l'approximation soit garantie quelle que soit la distribution D alors que dans le cadre de l'identification à la limite, l'identification doit se faire pour toute présentation.

7.2.6 Résultats PAC pour les langages réguliers

Il existe nombreux résultats négatifs concernant l'identification PAC. Le résultat qui va nous intéresser principalement a été démontré par Kearns et Valiant [KV89] : ils ont prouvé que la PAC-identification polynomiale, indépendamment de la représentation²² de la classe des automates finis déterministes acycliques de taille est un problème NP-difficile.

Pitt et Warmuth ont montré que le problème qui consiste à retrouver un automate polynomial plus grand que le plus petit DFA compatible avec les données était NP-difficile. Ils montrent de plus que la taille du vocabulaire prend part à la difficulté de l'inférence :

Théorème 7.4 ([PW93])

Le problème suivant est NP-difficile : étant donné une constante k et des exemples positifs et négatifs définis sur un alphabet Σ^* , $|\Sigma| = 2$, existe-t-il un DFA comportant au plus n^k états compatible avec les données, avec n la taille du plus petit automate compatible.

21. Le symbole \oplus désigne la différence symétrique de deux langages et $|\Sigma|$ est la taille de l'alphabet.

22. L'apprentissage indépendant de la représentation consiste à laisser à l'algorithme le choix de la représentation : AFN, AFD, grammaire, etc.

Si l'on considère que la taille de l'alphabet n'est pas donnée mais fait partie du problème, on a :

Théorème 7.5 ([PW93])

Quelle que soit la constante $\epsilon > 0$, il n'existe pas d'algorithme polynomial permettant de déterminer s'il existe un automate compatible avec les données de taille $n^{(1-\epsilon)\log\log n}$.

Les principales variations du cadre *PAC* portent sur des restrictions sur la classe des distributions des exemples. Le principe est de donner à l'algorithme de « bons » exemples. Ce peut être fait soit en définissant un professeur qui choisit les exemples en fonction de la cible, soit en imposant des contraintes sur la distribution de probabilités : l'algorithme est alors presque assuré de travailler avec des données caractéristiques de la cible. Deux idées qui vont dans ce sens ont été proposées par Denis et al. [DDG96] d'une part et par Denis et Gilleron [DG97] d'autre part. La première approche se conforme au principe qui stipule que les exemples les plus simples doivent être préférés. Le deuxième point de vue propose que l'enseignant choisisse des exemples afin de faciliter l'apprentissage. Nous allons présenter maintenant ces deux approches.

7.2.7 Apprentissage *PACS*: *PAC Simple*

Le modèle *PACS* (*PAC Simple*) proposé par Denis et al. [DDG96] a pour but de donner à l'algorithme d'apprentissage des exemples qui sont en adéquation avec la cible. Ils définissent donc un professeur qui construit un ensemble d'apprentissage en fonction de la cible. Un autre professeur « adversaire » ou arbitre ajoute ensuite des exemples de son choix à l'ensemble d'apprentissage afin d'éviter une collusion complète entre l'enseignant et l'apprenant. Le système demeure cependant d'une certaine manière potentiellement collusif. Dans ce modèle, étant donné une cible r , les exemples donnés à l'algorithme d'apprentissage, (c'est-à-dire les exemples simples) sont ceux qui ont une complexité de Kolmogorov étant donné r faible. Plus formellement, si on note $K(\alpha|r)$ la complexité de Kolmogorov²³ du mot α étant donné r , l'ensemble d'apprentissage défini pour un DFA r sera : $S_{simple}^r = \{\alpha \in \Sigma^* : K(\alpha|r) < \mu \log(|r|)\}$, où μ est une constante.

Résultats

Denis et al. [DDG96] ont montré que dans ce modèle les langages k -réversibles (voir au paragraphe 7.4.1) étaient identifiables.

Par ailleurs, Parekh et Honavar [PH97] ont montré que les DFA de taille N , N connu et fixé, étaient probablement exactement identifiables dans le modèle *PACS*. Ce résultat est plus fort que la *PACS*-apprenabilité au sens où l'identification de l'automate cible peut se faire avec une grande probabilité. Plus précisément, ils ont montré que l'échantillon caractéristique nécessaire à l'identification à la limite de l'algorithme RPNI²⁴ était présent avec une probabilité supérieure à $1 - \delta$, lorsque l'on se place dans le cadre *PACS* ayant pour critère de confiance δ .

La faiblesse principale de ce cadre de travail est qu'il suppose la possibilité du calcul de la complexité de Kolmogorov (voir le chapitre 17). C'est donc un cadre purement théorique. Sa deuxième faiblesse est la difficulté d'interdire une collusion entre oracle et apprenant, c'est-à-dire le codage de l'hypothèse dissimulé dans les exemples. Conscients de ce problème, certains des auteurs se sont tournés vers une approche plus réaliste en pratique.

23. Pour une étude de la complexité de Kolmogorov, le lecteur pourra consulter [LV97].

24. Il sera question en détail de cet algorithme au paragraphe 7.5.4.

7.2.8 Apprentissage *PAC* avec distributions bienveillantes

Denis et Gilleron ont proposé [DG97] une adaptation de cadre *PAC*: l'apprentissage *PAC* avec des distributions bienveillantes.

Cette approche affaiblit le cadre *PAC* en opérant une restriction sur l'ensemble des distributions de probabilités selon lesquelles les exemples sont tirés. La classe de distributions bienveillantes est définie en fonction de la cible que l'algorithme doit identifier.

Informellement, un *exemple utile* est un exemple dont la taille est polynomiale en la taille de la cible. Une distribution bienveillante est une distribution telle que tous les *exemples utiles* ont une probabilité non nulle. Par ailleurs, on peut supposer que plus les exemples auront une probabilité faible, plus l'inférence sera difficile à réaliser. C'est pourquoi les auteurs ajoutent un paramètre de complexité au modèle *PAC*: l'algorithme d'inférence devra être polynomial en l'inverse de plus petite probabilité des exemples rencontrés.

Résultats

Denis et Gilleron montrent [DG97] d'une part que la classe des DFA définis sur Σ^n , n fixé et connu, est *PAC*-identifiable sous distributions bienveillantes.

La preuve est basée sur la propriété d'identification à la limite de l'algorithme RPNI [OG92a]. Elle consiste à construire un échantillon d'apprentissage à partir d'un enseignant et à montrer qu'un tel ensemble d'apprentissage contient probablement l'échantillon caractéristique.

Par ces deux approches, on appréhende maintenant mieux la difficulté de définir la notion d'apprentissage d'un automate. Le problème consiste à poser des contraintes sur les exemples que l'on donne à l'apprenant et à travers lui, sur la distribution de probabilités sur les exemples.

Si on ne fait aucun présupposé sur la pertinence des exemples (cadre *PAC* classique), le cadre est trop contraignant et des algorithmes qui infèrent correctement en pratique n'apprennent pas dans ce cadre. Si on propose de donner des exemples pertinents (cadre *PACS* et cadre *PAC* sous distributions bienveillantes) on ne peut éviter le problème de la collusion.

7.3 L'espace de recherche de l'inférence régulière

7.3.1 Le point de la situation

Nous avons défini l'inférence régulière comme la recherche d'un automate A inconnu à partir duquel un échantillon positif I_+ est supposé avoir été généré. Étant donnée l'hypothèse supplémentaire de complétude structurelle de I_+ relativement à l'automate A , nous allons maintenant préciser la définition de ce problème comme une exploration dans un ensemble structuré construit sur I_+ .

Nous revenons maintenant à un protocole de présentation tout à fait classique: l'algorithme d'apprentissage dispose de l'ensemble des exemples positifs et négatifs et tout renseignement supplémentaire par un oracle est interdit.

Nous avons vu au paragraphe 7.1.1 un certain nombre de définitions et de notions formelles. Rappelons-les rapidement avant de les utiliser pour construire cet ensemble structuré.

- À partir d'un échantillon positif I_+ , on peut construire deux automates particuliers qui n'acceptent que lui: $MCA(I_+)$ et $PTA(I_+)$. Le second est un automate dérivé du premier.

- À partir d'un automate A quelconque, on peut construire d'autres automates par l'opération de dérivation : chaque état d'un automate dérivé de A est la fusion d'un certain nombre d'états de A .
- Un automate dérivé de A reconnaît un langage qui inclut $L(A)$.
- L'ensemble des automates dérivés d'un automate A forme le treillis $\text{Lat}(A)$.
- Un échantillon est structurellement complet par rapport à un automate s'il est représentatif du langage qu'il accepte.

Nous disposons maintenant de tout le matériel technique pour définir l'ensemble de toutes les solutions d'un problème d'inférence régulière.

7.3.2 Deux propriétés fondamentales

Un premier théorème nous assure que, sous le biais de la complétude structurelle, l'ensemble des hypothèses compatibles avec l'échantillon est exactement le treillis construit sur $MCA(I_+)$.

Théorème 7.6

Soit I_+ un échantillon positif d'un langage quelconque régulier L et soit A n'importe quel automate acceptant exactement L . Si I_+ est structurellement complet relativement à A alors A appartient à $\text{Lat}(MCA(I_+))$. Réciproquement, si un automate A appartient à $\text{Lat}(MCA(I_+))$ alors I_+ est structurellement complet relativement à A .

Le second théorème assure que l'on peut réduire l'espace de recherche si on cherche l'automate canonique d'un langage.

Théorème 7.7

Soit I_+ un échantillon positif d'un quelconque langage régulier L et soit $A(L)$ l'automate canonique acceptant L . Si I_+ est structurellement complet relativement à $A(L)$ alors $A(L)$ appartient à $\text{Lat}(PTA(I_+))$.

De plus, nous pouvons énoncer une propriété d'inclusion entre ces deux treillis :

Propriété 7.5

$$\text{Lat}(PTA(I_+)) \subseteq \text{Lat}(MCA(I_+))$$

Cette propriété découle directement de la définition 7.16 du $PTA(I_+)$ qui est un automate quotient du $MCA(I_+)$. De plus, comme le treillis $\text{Lat}(PTA(I_+))$ est généralement strictement inclus dans le treillis $\text{Lat}(MCA(I_+))$, rechercher une solution dans $\text{Lat}(PTA(I_+))$ au lieu de $\text{Lat}(MCA(I_+))$ permet de considérer un espace de recherche plus restreint.

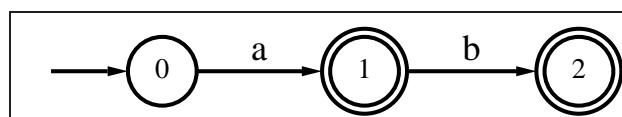


FIG. 7.13 – Un automate pour lequel l'échantillon $\{ab\}$ n'est pas structurellement complet.

En guise d'illustration, considérons l'exemple suivant. Conformément à la définition 7.14, l'échantillon $I_+ = \{ab\}$ n'est pas structurellement complet relativement à l'automate A de la figure 7.13, où q_1 et q_2 sont des états d'acceptation. L'état 1, qui est final, n'est l'état d'acceptation d'aucune phrase de I_+ . Pour cette raison, l'automate A ne peut pas être dérivé du

$MCA(I_+)$ et évidemment pas non plus du $PTA(I_+)$. Il n'est pas possible de définir une partition π de l'ensemble des états Q_{MCA} telle que A corresponde à $MCA(I_+)/\pi$ avec $q_1 \in F_{MCA/\pi}$. Au contraire, l'échantillon $\{ab, a\}$ est structurellement complet relativement à l'automate A qui peut donc être dérivé du MCA , ou du PTA associés à cet échantillon.

7.3.3 La taille de l'espace de recherche

Grâce aux résultats présentés à la section 7.3.2, nous savons que si nous disposons d'un échantillon positif I_+ d'un langage inconnu L , avec I_+ structurellement complet relativement à un automate inconnu A acceptant exactement L , alors nous pouvons dériver A pour une certaine partition π de l'ensemble des états du $MCA(I_+)$. Nous pouvons dès lors considérer l'inférence régulière comme un problème de recherche de la partition π .

Mais l'exploration de cet espace n'est pas un problème facile: sa taille est le nombre de partitions $|\mathcal{P}(N)|$ d'un ensemble à N éléments où N est ici le nombre d'états de l'élément nul du treillis, c'est-à-dire $PTA(I_+)$ ou $MCA(I_+)$. Ce nombre croît plus qu'exponentiellement en fonction de N . À titre d'exemple, $|\mathcal{P}(10)| = 10^5$, $|\mathcal{P}(20)| = 510^{13}$, $|\mathcal{P}(30)| = 8.5 \times 10^{23} \dots$

7.4 L'inférence régulière sans échantillon négatif

Étant donné un échantillon positif I_+ d'un langage régulier L , nous avons à définir le critère qui guidera la recherche de l'automate A acceptant I_+ . En particulier, le $MCA(I_+)$ et le $PTA(I_+)$ acceptent tous deux l'échantillon positif. Cependant, ils n'acceptent aucune autre phrase du langage L et, donc, ne généralisent pas l'information contenue dans les données d'apprentissage.

Nous savons, par la propriété 7.1, que n'importe quel automate dérivé du $MCA(I_+)$ accepte un langage incluant I_+ . En ce sens, n'importe quel automate appartenant à $Lat(MCA(I_+))$ constitue une généralisation possible de l'échantillon positif. Conformément au résultat de Gold présenté à la section 7.2.2, nous savons qu'il n'est pas possible d'identifier la classe entière des langages réguliers à partir d'un échantillon positif seulement. En particulier, aucun exemple positif ne peut éviter une surgénéralisation éventuelle de l'inférence. En d'autres termes, si une solution proposée par un algorithme d'inférence correspond à un langage incluant strictement le langage à identifier, aucun exemple positif ne peut contredire cette solution.

Pour éviter le risque de surgénéralisation, deux approches sont classiquement utilisées. La première possibilité consiste à rechercher la solution dans une sous-classe particulière des langages réguliers ; il s'agit alors d'une méthode *caractérisable*. La seconde possibilité consiste à utiliser une information *a priori* pour guider la généralisation recherchée, le propre des méthodes *heuristiques*.

7.4.1 Une méthode caractérisable : l'inférence de langages k-réversibles

Angluin a proposé une méthode d'inférence qui identifie à la limite à partir d'une présentation positive la classe des langages *k-réversibles*, pour k quelconque mais fixé a priori [Ang82a]. Nous détaillons ci-après les principales notions nécessaires à la définition de cette méthode d'inférence.

Définition 7.24

L'automate inverse $A^r = (Q, \Sigma, \delta^r, F, I)$ d'un automate $A = (Q, \Sigma, \delta, I, F)$ est défini comme suit :

$$\forall q \in Q, \quad \forall a \in \Sigma, \quad \delta^r(q, a) = \{q' \in Q \mid q \in \delta(q', a)\}.$$

Par conséquent, l'automate inverse A^r est obtenu en inversant les états²⁵ d'entrée et de sortie de l'automate original A , et en inversant également le sens des transitions de A . Nous présentons à la figure 7.14 un exemple d'automate et son inverse.

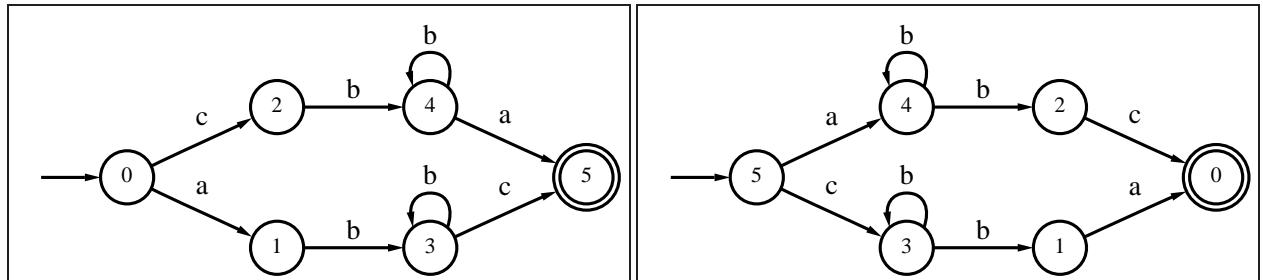


FIG. 7.14 – L'automate A et son inverse A^r .

Définition 7.25

Un automate $A = (Q, \Sigma, \delta, q_0, F)$ est déterministe avec anticipation k si²⁶

$$\forall q \in Q, \forall x \in \Sigma^*, |x| > k, \quad |\delta^*(x, q)| \leq 1.$$

Par conséquent, un automate est déterministe avec anticipation k si, lors de l'acceptation d'une chaîne quelconque, il existe au plus un état auquel cette acceptation peut mener en anticipant de k lettres par rapport à la position courante dans la chaîne. Un AFD est déterministe avec anticipation 0. L'automate A^r de la figure 7.14 est déterministe avec anticipation 1.

Définition 7.26

Un automate A est k -réversible s'il est déterministe et si son inverse A^r est déterministe avec anticipation k .

Définition 7.27

Un langage L est k -réversible s'il existe un automate k -réversible qui l'accepte.

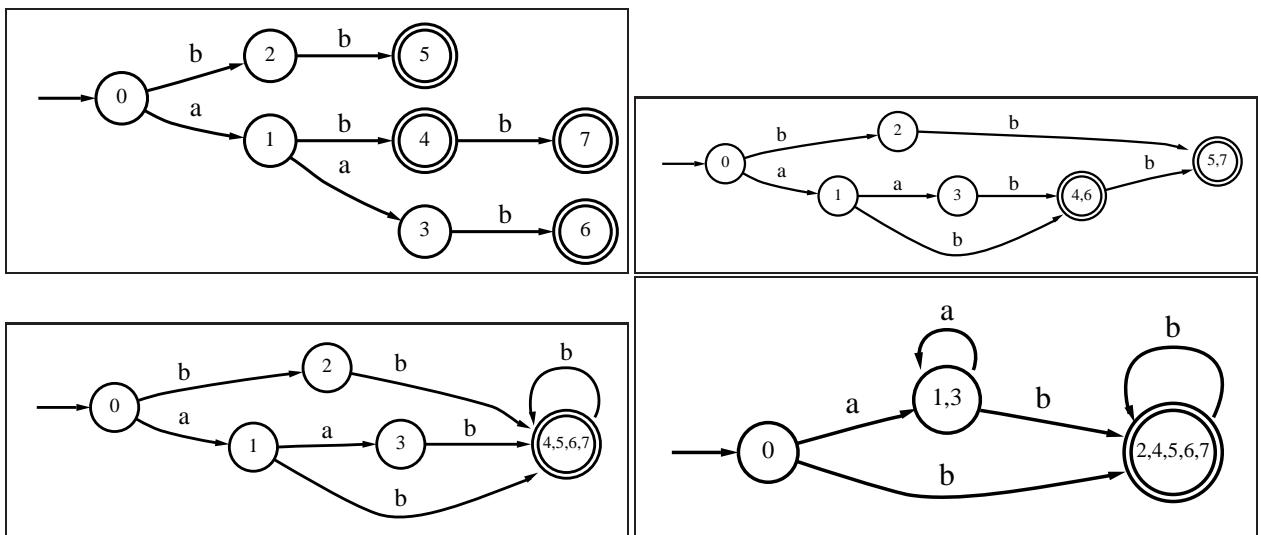
L'algorithme k -RI²⁷ consiste à partir de l'accepteur des préfixes de l'échantillon positif puis à fusionner des états, c'est-à-dire à définir un nouvel automate quotient, tant que la condition de k -réversibilité est violée. Angluin [Ang82a] a démontré que cet algorithme permettait d'identifier la classes des langages k -réversibles et qu'il produisait, pour I_+ et k fixé, l'automate canonique du plus petit langage k -réversible incluant I_+ . Nous présentons à la figure 7.15 un exemple d'exécution de l'algorithme k -RI avec $I_+ = \{ab, bb, aab, abb\}$ et $k = 1$.

La fonction k -réversible (A/π) renvoie *VRAI* si l'automate quotient A/π est k -réversible, elle renvoie *F AUX* sinon. La fonction *non-réversible* ($A/\pi, \pi$) renvoie deux blocs distincts de π , tels que la condition de k -réversibilité ne soit pas vérifiée pour les deux états associés dans A/π . La fonction *dériver* (A, π') renvoie l'automate quotient de A conformément à la partition π' .

25. Nous avons défini à la section 7.1, les automates finis comme comprenant un seul état d'entrée et un ensemble d'états de sortie. Ces automates permettent de représenter un langage régulier quelconque. La généralisation aux automates comportant un ensemble d'états d'entrée est triviale.

26. δ^* désigne l'extension classique à $\Sigma^* \times Q \rightarrow 2^Q$, de la fonction de transition δ .

27. L'abréviation k -RI provient de la terminologie anglaise pour *k-Reversible Inference*.

FIG. 7.15 – Exemple d'exécution de l'algorithme $k\text{-RI}$, avec $I_+ = \{ab, bb, aab, abb\}$.**Algorithm 7.1 Algorithme $k\text{-RI}$**

Entrée: k , l'ordre du modèle, I_+ , l'échantillon positif

Sortie: A_k , un automate canonique acceptant le plus petit langage k -réversible incluant I_+
 $\{N\}$ désigne le nombre d'états de $PTA(I_+)$

$\pi \leftarrow \{\{0\}, \{1\}, \dots, \{N-1\}\}$ {Un bloc par état du $PTA(I_+)$ }

$A \leftarrow PTA(I_+)$

tant que $\neg(k\text{-réversible } (A/\pi))$ **faire**

$(B_1, B_2) \leftarrow \text{non-réversible } (A/\pi, \pi)$

$\pi \leftarrow \pi \setminus \{B_1, B_2\} U \{B_1 U B_2\}$ {Fusion du bloc B_1 et du bloc B_2 }

fin tant que

A/π

Rappelons qu'il est prouvé que cette méthode identifie à la limite la classe des langages k -réversibles pour k donné.

7.4.2 Une méthode heuristique : l'algorithme ECGI

L'algorithme ECGI est une méthode d'inférence heuristique conçue pour extraire l'information sur la longueur de sous-chaînes des éléments de I_+ et de la concaténation de ces sous-chaînes. Cet algorithme est incrémental, c'est-à-dire qu'il mesure une « distance » entre un nouvel exemple et le modèle existant et adapte le modèle conformément. Le langage accepté par la grammaire inférée inclut I_+ ainsi que d'autres chaînes obtenues par concaténation de sous-chaînes des éléments de I_+ .

Nous reprenons la description de l'algorithme ECGI tel qu'il a été formellement présenté dans [RV88, RPV89a]. La représentation utilisée est celle des grammaires régulières. Nous illustrons également son fonctionnement à l'aide de la représentation équivalente en automates.

Propriété 7.6

L'algorithme ECGI produit une grammaire $G = (N, \Sigma, P, S)$ régulière, non déterministe, sans

cycle²⁸ et qui vérifie la condition suivante :

$$\forall A, B, C \in N, \quad \forall b, a \in \Sigma \quad \text{si } (B \rightarrow aA) \in P \text{ et } (C \rightarrow bA) \in P \text{ alors } b = a.$$

En d'autres termes, la même lettre de l'alphabet terminal est associée avec toutes les productions ayant le même non-terminal en partie droite. Cela permet d'associer les terminaux aux états plutôt qu'aux arcs de l'automate équivalent.

Définition 7.28

À toute production dans P , sont associés les règles d'erreur suivantes :

$$\text{Insertion de } a : \quad A \rightarrow aA, \forall(A \rightarrow bB) \in P, \forall a \in \Sigma$$

$$\text{Substitution de } b \text{ par } a : \quad A \rightarrow aB, \forall(A \rightarrow bB) \in P, \forall a \in \Sigma$$

$$A \rightarrow a, \forall(A \rightarrow b) \in P, \forall a \in \Sigma$$

$$\text{Suppression de } b : \quad A \rightarrow B, \forall(A \rightarrow bB) \in P, \forall a \in \Sigma$$

$$A \rightarrow \epsilon, \forall(A \rightarrow b) \in P, \forall a \in \Sigma$$

Définition 7.29

La grammaire étendue $G' = (N', \Sigma, P', S)$ de G est la grammaire obtenue en ajoutant les règles d'erreurs à P .

Définition 7.30

La dérivation corrective optimale de $\beta \in \Sigma^*$ est la dérivation de β , conformément à la grammaire G' et qui utilise un nombre minimal de règles d'erreurs.

L'algorithme ECGI construit d'abord la grammaire canonique acceptant la première chaîne de I_+ . Ensuite, pour chaque nouvel exemple β , la dérivation corrective optimale est calculée et la grammaire est étendue conformément à cette dérivation²⁹. Elle produit la séquence optimale de règles correctes (celles déjà présentes dans la grammaire) et de règles d'erreur pour l'acceptation de β . Nous présentons à la figure 7.16 un exemple d'exécution de l'algorithme ECGI prenant en entrée l'échantillon $I_+ = \{aabb, abbb, abbab, bbb\}$ et en représentant chaque grammaire par un automate associé. Les états et transitions en pointillés correspondent à l'extension de la grammaire à chaque étape.

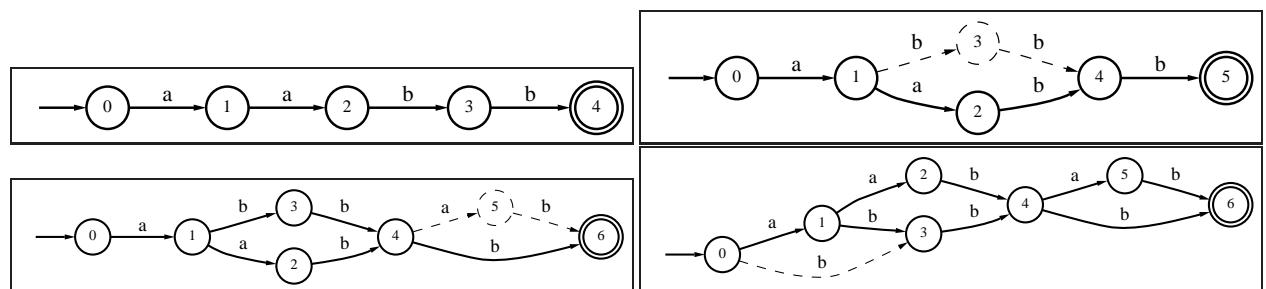


FIG. 7.16 – Exemple d'exécution de l'algorithme ECGI, avec $I_+ = \{aabb, abbb, abbab, bbb\}$.

28. Une grammaire est sans cycle si quel que soit le non terminal A , aucune phrase engendrée à partir de A n'est de la forme xA , avec $x \in \Sigma^+$. En pratique, cela revient à dire que l'automate correspondant ne comporte pas de boucle et donc que le langage engendré est fini.

29. Cette dérivation est obtenue par une procédure d'analyse corrective qui constitue une application de l'algorithme de programmation dynamique (voir le chapitre 3 pour les généralités et le chapitre 13 pour une application particulière de la programmation dynamique: l'algorithme de Viterbi).

Algorithme 7.2 Algorithme ECGI

Données d'entrée : I_+

Sortie: Une grammaire G compatible avec I_+ et vérifiant la propriété 7.6

```

 $x \leftarrow I_+^1 ; n \leftarrow |x|$ 
 $N \leftarrow \{A_0, \dots, A_{n-1}\} ; \Sigma \leftarrow \{a_1, \dots, a_n\}$ 
 $P \leftarrow \{(A_i \rightarrow a_i A_i), i = 1, \dots, n-1\} \cup \{A_{n-1} \rightarrow a_n\}$ 
 $S \leftarrow A_0 ; G_1 \leftarrow (N, \Sigma, P, S)$ 
pour  $i = 2$  à  $|I_+|$  faire
   $G \leftarrow G_{i-1} ; x \leftarrow I_+^i ; P' \leftarrow \text{dériv\_optim}(x, G_{i-1})$ 
  pour  $j = 1$  à  $P'$  faire
     $G \leftarrow \text{étendre\_gram}(G, p_j)$ 
  fin pour
   $G_i \leftarrow G$ 
fin pour
Retourner  $G$ 

```

La fonction $\text{dériv_optim}(x, G_{i-1})$ renvoie la dérivation corrective optimale de la chaîne x conformément à la grammaire G_{i-1} . La fonction $\text{étendre_gram}(G, p_j)$ renvoie la grammaire étendue conformément à la règle d'erreur p_j .

Supposons que nous disposions désormais d'un échantillon négatif I_- du langage inconnu L . Dans ce cas, nous savons que nous pouvons, en principe, identifier à la limite au moins n'importe quel langage régulier (voir section 7.2.2). L'inférence peut être alors considérée comme la découverte d'un automate A compatible avec les échantillons positif et négatif, c'est-à-dire tel que $I_+ \subseteq L(A)$ et $I_- \cap L(A) = \emptyset$. Il existe un grand nombre d'automates compatibles appartenant à $\text{Lat}(\text{MCA}(I_+))$. $\text{MCA}(I_+)$ satisfait ces conditions mais ne généralise pas l'échantillon positif. Par conséquent, nous pouvons chercher une solution plus générale *sous le contrôle* de l'échantillon négatif.

Si nous choisissons la simplicité de l'automate inféré comme critère de généralité et que nous restreignons la recherche à des automates déterministes, la solution cherchée est alors l'AFD compatible et comportant le nombre minimal d'états. Il s'agit du problème du *plus petit AFD compatible*, déjà évoqué plus haut (paragraphe 7.2.3). Nous avons vu qu'il n'existe pas d'algorithme polynomial résolvant ce problème dans tous les cas. Cependant, on peut trouver un algorithme polynomial qui permet, en général, de trouver une solution quasi exacte. Nous détaillons cet algorithme à la section 7.5.4. Deux autres algorithmes seront abordés ensuite. Afin de comparer ces trois algorithmes et d'étudier le problème du plus petit AFD compatible, nous poursuivons la caractérisation de l'espace de recherche introduite à la section 7.3.

7.5 L'inférence régulière sous contrôle d'un échantillon négatif

7.5.1 L'ensemble frontière

La notion d'ensemble frontière permet de définir un sous-espace intéressant de solutions à un problème d'inférence avec contre-exemples.

Définition 7.31

Une antichaîne \overline{AS} dans un treillis d'automates est un ensemble d'automates tel qu'aucun élément de \overline{AS} n'est relié par la relation de dérivation avec un autre élément de \overline{AS} .

Définition 7.32

Un automate A est à une profondeur maximale dans un treillis d'automates s'il n'existe pas, dans ce treillis, d'automate A' , différent de A , tel que A' puisse être dérivé de A et tel que $L(A') \cap I_- = \emptyset$.

Définition 7.33

L'ensemble frontière $BS_{MCA}(I_+, I_-)$ est l'antichaîne dont chaque élément est à une profondeur maximale dans $\text{Lat}(MCA(I_+))$.

Propriété 7.7

L'ensemble frontière $BS_{PTA}(I_+, I_-)$ contient l'automate canonique $A(L)$ de tout langage régulier L dont I_+ est un échantillon positif et I_- un échantillon négatif.

Par conséquent, l'ensemble frontière du treillis construit sur $MCA(I_+)$ est l'ensemble des automates les plus généraux compatibles avec l'échantillon positif et l'échantillon négatif. D'autre part, le problème du plus petit AFD compatible se ramène à découvrir le plus petit AFD appartenant à l'ensemble frontière du treillis construit sur $PTA(I_+)$.

7.5.2 Le lien avec l'espace des versions

Compte tenu de sa construction à partir de I_+ puis de son élagage par I_- , le treillis des solutions est un cas particulier de l'espace des versions décrit au chapitre 4. On peut considérer simplement que le $PTA(I_+)$ est l'unique généralisation la plus spécifique, donc que $S = \{PTA(I_+)\}$. En revanche, il existe plusieurs solutions plus générales, qui ne sont autres que les éléments de l'ensemble frontière : $G = BS_{MCA}(I_+, I_-)$. Tout élément du treillis qui peut se transformer en élément de BS par fusion d'états est une solution. L'inférence grammaticale ajoute un biais pour trouver une des solutions de l'espace des versions : chercher un automate sur BS , donc tel que toute fusion de deux de ses états conduise à l'acceptation d'exemples négatifs. C'est donc une heuristique de simplicité qui est adoptée, conformément au principe du rasoir d'Occam (voir au chapitre 3 le paragraphe 3.4.7) et au principe *MDL*.

7.5.3 Les algorithmes RIG et BRIG

Étant donné des échantillons positif I_+ et négatif I_- , il est facile, par un algorithme nommé RIG³⁰ [MdG94], de construire complètement $BS_{MCA}(I_+, I_-)$. Cet algorithme procède par énumération des automates dérivés du $MCA(I_+)$, c'est-à-dire des partitions dans $\text{Lat}(MCA(I_+))$. Il s'agit d'une énumération en largeur à partir de l'élément nul du treillis des partitions, le

30. L'abréviation RIG provient de la terminologie anglaise pour *Regular Inference of Grammars*.

$MCA(I_+)$. Cette énumération ne conserve que les automates compatibles à chaque profondeur dans le treillis. Par la propriété 7.1 d'inclusion des langages, nous pouvons effectuer un élagage par héritage, qui consiste à éliminer tous les automates à la profondeur $i + 1$ qui dérivent d'au moins un automate non compatible à la profondeur i . Ensuite, les automates restant à la profondeur $i + 1$ subissent un *élagage direct* qui consiste à éliminer les automates non compatibles à cette profondeur. Finalement, l'ensemble frontière est obtenu en mémorisant tous les automates compatibles et qui ne possèdent aucun dérivé compatible. La solution proposée par l'algorithme RIG est, par exemple, le premier automate déterministe rencontré à la profondeur maximale de l'ensemble frontière, mais l'algorithme RIG peut également fournir comme solution tout l'ensemble frontière (comme le ferait l'algorithme de l'élimination des candidats dans la méthode de l'espace des versions).

La complexité de RIG étant non polynomiale³¹, une version heuristique, nommée BRIG, a été proposée [MdG94]. Elle consiste à ne considérer qu'une petite proportion des partitions, générée aléatoirement. Cette sélection aléatoire permet de garantir, en pratique, une complexité polynomiale et de construire un sous-ensemble du BS . L'algorithme BRIG est clairement de nature heuristique car il n'existe pas de caractérisation de la classe de langages qu'il identifie (son caractère aléatoire implique que différentes exécutions partant des mêmes données ne conduisent pas nécessairement au même résultat).

7.5.4 L'algorithme RPNI

7.5.4.1 Une exploration efficace du treillis

L'algorithme RPNI [OG92b] effectue une recherche en profondeur dans $Lat(PTA(I_+))$ et trouve un optimum local au problème du plus petit AFD compatible. Nous savons que l'identification d'un langage régulier L par exploration du treillis des partitions du $PTA(I_+)$ n'est envisageable que si l'on suppose la complétude structurelle de I_+ relativement à l'automate canonique $A(L)$. Sous cette hypothèse, l'algorithme RPNI est particulièrement efficace.

Par construction du $PTA(I_+)$ (voir définition 7.16), chacun de ses états correspond à un préfixe unique et les préfixes peuvent être triés par ordre lexicographique³² $<$. Cet ordre s'applique donc également aux états du $PTA(I_+)$. L'algorithme RPNI procède en $N - 1$ étapes où N est le nombre d'états du $PTA(I_+)$. La partition à l'étape i est obtenue en fusionnant les deux premiers blocs, par ordre lexicographique, de la partition à l'étape $i - 1$ et qui, de plus, donne lieu à un automate quotient compatible.

L'automate quotient A/π' peut être non déterministe. La fonction *déterm-fusion* (A/π') réalise la *fusion pour déterminisation* en renvoyant la partition π'' obtenue en fusionnant récursivement tous les blocs de π' qui créent le non-déterminisme³³. Si A/π' est déterministe, la partition π'' est égale à la partition π' .

Si l'automate déterministe ainsi obtenu est correct pour I_- , c'est-à-dire s'il n'en accepte aucun échantillon, alors *RPNI* est relancé à partir de cette solution provisoire. Pour conserver

31. Dans le pire des cas, aucun élagage par I_- n'est effectué. Dans ce cas, l'algorithme RIG explore tout le treillis des partitions. Malgré un élagage effectif par l'échantillon négatif, il a été observé expérimentalement que la complexité de RIG reste non polynomiale en fonction de N , le nombre d'états du $MCA(I_+)$ [MdG94].

32. L'ordre lexicographique sur les chaînes de Σ^* , correspond à un ordre par longueur des chaînes et, pour une longueur donnée, à l'ordre alphabétique. Par exemple, pour l'alphabet $\{a, b\}$, les premières chaînes dans l'ordre lexicographique sont : $\epsilon, a, b, aa, ab, ba, bb, aaa, aab\dots$

33. L'opération de fusion pour déterminisation d'un AFN ne doit pas être confondue avec l'algorithme classique de déterminisation d'un AFN, qui a pour objet de produire un AFD acceptant le même langage [AU72]. Conformément à la propriété 7.1, l'AFD obtenu par fusion des états d'un AFN accepte, en général, un surlangage de l'AFN dont il provient.

Algorithme 7.3 Algorithme RPNI

```

Entrée:  $I_+, I_-$ 
Sortie: Une partition du  $PTA(I_+)$  correspondant à un AFD compatible avec  $I_+$  et  $I_-$  { $N$  désigne le nombre d'états de  $PTA(I_+)$ }
 $\pi \leftarrow \{\{0\}, \{1\}, \dots, \{N - 1\}\}$ 
 $A \leftarrow PTA(I_+)$ 
pour  $i = 1$  à  $N - 1$  faire
  pour  $j = 0$  à  $i - 1$  faire
     $\pi' \leftarrow \pi \setminus \{B_j, B_i\} U \{B_i \cup B_j\}$ ; {Fusion du bloc  $B_i$  et du bloc  $B_j$ }
```

$\pi'' \leftarrow \text{déterm-fusion } (A / \pi')$

si A / π'' correct(I_-) **alors**

$\pi \leftarrow \pi'';$

fin si

fin pour

fin pour

Retourner $A \leftarrow A / \pi$;

l'ordre lexicographique, quand un état est fabriqué par fusion et correspond donc à un bloc d'états du PTA , il prend alors le rang de l'état du PTA de rang le plus faible dans ce bloc.

La solution proposée par l'algorithme RPNI est un automate déterministe appartenant au $BS_{PTA}(I_+, I_-)$. Par la propriété 7.5.1, nous savons qu'il s'agit de l'automate canonique pour le langage qu'il accepte. Cependant, il ne s'agit du plus petit AFD compatible que si les données d'apprentissage satisfont une condition supplémentaire, c'est-à-dire contiennent un échantillon dit *caractéristique*, formellement défini dans [OG92b]. En d'autres termes, lorsque les données d'apprentissage sont suffisamment représentatives, la découverte de l'automate canonique du langage à identifier est garantie. De plus, cet automate est également la solution du problème du plus petit AFD compatible, dans ce cas particulier. Les auteurs ont démontré que la taille d'un échantillon caractéristique propre à cet algorithme est $\mathcal{O}(n^2)$, où n est le nombre d'états de l'automate cible [OG92b]. La complexité calculatoire de l'algorithme RPNI, dans sa dernière version publiée, est $\mathcal{O}((|I_+| + |I_-|) \cdot |I_+|^2)$.

Il est de plus démontré [TB73b] que si l'échantillon d'apprentissage contient toutes les chaînes de longueur inférieures à $2k - 1$ où k est le nombre des états de l'automate cible alors, l'identification est garantie. Mais cette propriété est fine: si l'ensemble d'apprentissage contient toutes les chaînes sauf une partie infime de l'ensemble caractéristique, alors, l'identification n'est plus garantie [Ang78b].

7.5.4.2 Un exemple

Partons des ensembles $I_+ = \{\epsilon, ab, aaa, aabaa, aaaba\}$ et $I_- = \{aa, baa, aaab\}$. L'automate $PTA(I_+)$ est représenté sur la figure 7.17, en haut. Il possède dix états.

- L'algorithme *RPNI* commence par fusionner deux états. En l'absence d'autre indication les états 0 et 1 sont choisis. Ceci conduit à un indéterminisme, puisque l'état $\{0, 1\}$ mène désormais à lui-même et à l'état 2 par une transition a . La procédure de fusion pour déterminisation est donc appellée.
 - Elle fusionne d'abord l'état $\{0, 1\}$ avec l'état 2. Mais ceci amène deux autres indéter-

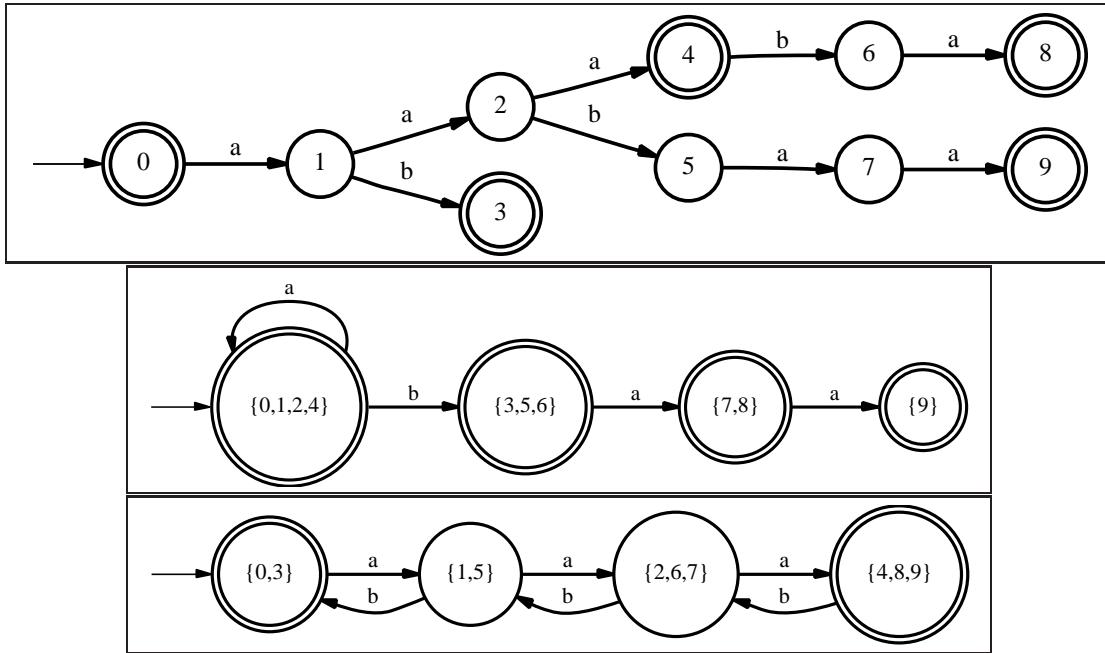


FIG. 7.17 – Exemple d'exécution de l'algorithme RPNI. En haut, $PTA(I_+)$, avec $I_+ = \{\epsilon, ab, aaa, aabaa, aaaba\}$. Au centre, le résultat de la fusion des états 0 et 1 suivie de la procédure de fusion pour déterminisation. Cet automate n'est pas une solution puisqu'il accepte au moins une phrase (ici, les trois) de $I_- = \{aa, baa, aaab\}$. En bas, l'automate obtenu comme solution.

minismes: $\{0, 1, 2\}$ mène à 4 et à lui-même par a , mais aussi à 5 et 3 par b .

- Elle poursuit en fusionnant $\{0, 1, 2\}$ avec 4 et en fusionnant 5 avec 3. Il reste un nouvel indéterminisme: $\{0, 1, 2, 4\}$ mène par b à $\{3, 5\}$ et 6.
- Elle fusionne derechef ces deux derniers états, ce qui mène à un automate déterministe (figure 7.17, au milieu).

Cet automate accepte la phrase aa dans I_- . On doit donc remettre en question la fusion initiale des états 0 et 1.

- L'étape suivante est de fusionner les états 0 et 2, puis de procéder à la fusion par déterminisation. Ici aussi, l'automate obtenu accepte une phrase de I_- .
- En revanche, l'étape suivante, la fusion de 0 et 3, mène à un automate cohérent avec I_+ et I_- , dont on sait qu'il est sur le BS. C'est cette solution qui est retenue (figure 7.17, en bas).

Le langage reconnu peut s'exprimer comme « l'ensemble des phrases dont le nombre de a moins le nombre de b vaut 0 modulo 3 ». L'échantillon a été choisi comme l'un des plus petits possibles qui soit caractéristique de ce langage, ce qui explique qu'on puisse l'inférer avec si peu d'information.

7.5.5 Variantes et extensions

La plupart des algorithmes efficaces d'inférence régulière sont actuellement des variations sur RPNI. La faiblesse de celui-ci est en effet d'imposer un ordre arbitraire aux essais successifs des fusion d'états du PTA (ceux-ci sont rangés dans l'ordre lexicographique du préfixe correspondant dans le PTA). Les algorithmes plus efficaces fusionnent de préférence les deux états les plus

« prometteurs » avant d'appliquer la fusion pour déterminisation. Ce terme est évidemment empirique, mais il existe des heuristiques expérimentalement efficaces pour le définir. On trouvera les détails dans [LPP98].

Un certain nombre d'autres possibilités ont été développées, comme le travail dans un espace d'automates non déterministes, ou l'inférence par fission (en partant de l'automate universel).

Citons également ici, pour anticiper sur le chapitre 8 un algorithme heuristique, nommé GIG³⁴, qui optimise par un *algorithme génétique* la recherche d'une solution optimale, c'est-à-dire à la profondeur maximale. Il fait évoluer dans $\text{Lat}(\text{PTA}(I_+))$ une population d'automates dérivés du $\text{PTA}(I_+)$, par le biais d'opérateurs spécifiques de mutation et de croisement. Une technique analogue est aussi évoquée dans ce même chapitre : les *stratégies d'évolution*, qui gèrent des populations d'automates.

7.6 L'inférence de grammaires algébriques.

7.6.1 Présentation

Comme on l'a vu aux paragraphes précédents, l'inférence des grammaires régulières est un problème bien formalisé, possédant (si l'on admet le biais de la complétude structurelle) un cadre algorithmique clair. Ce n'est pas le cas de l'apprentissage des grammaires algébriques. On peut donner deux raisons à cet état de fait :

- Les grammaires régulières sont un « tout petit » sous-ensemble des grammaires algébriques. L'espace de recherche d'un concept à partir d'exemples est donc beaucoup plus grand pour ces dernières.
- On ne dispose pas pour les grammaires algébriques d'une relation d'ordre qui permettrait de parcourir un treillis fini de solutions et de pratiquer un élagage systématique.

Ce second point est relié au fait que la représentation en automates finis, presque systématiquement utilisée pour l'inférence régulière, n'a pas d'équivalent commode pour les grammaires algébriques. Les algorithmes qui ont été proposés visent donc pour la plupart à contourner ces problèmes, soit en définissant des sous-classes de grammaires algébriques où l'on pourra appliquer des algorithmes du type de ceux utilisés pour les grammaires régulières, soit en enrichissant la présentation des exemples par un oracle (voir le chapitre 2). Nous présenterons brièvement une méthode de chaque type. La bibliographie de ce chapitre renvoie à des références plus complètes sur ce sujet difficile.

7.6.2 L'apprentissage à partir d'échantillons structurés.

Cette famille de méthodes [CR71], [Sak90] suppose que l'échantillon d'apprentissage (positif seulement) est composé non seulement de phrases, mais également pour chacune d'elles d'indications sur la façon dont la phrase a été engendrée par la grammaire à trouver. Cette hypothèse forte permet de mettre en œuvre un algorithme qui produit une grammaire compatible avec l'échantillon, en introduisant un nombre minimal de concepts ; en conséquence, l'espace de recherche est implicitement restreint et les grammaires trouvées appartiennent à une stricte sous-famille des langages algébriques.

Plus précisément, l'échantillon d'apprentissage est constitué de phrases structurées par un parenthésage représentant leur *arbre de dérivation* [AU72]. Par exemple, la grammaire G :

1. $S \rightarrow a,$

34. L'abréviation **GIG** provient de la terminologie anglaise pour *Grammatical Inference by Genetic search*.

$$2. \quad S \rightarrow a + S$$

engendre la phrase: $x = a + a + a$ par l'arbre de dérivation de la figure 7.18. Sous forme

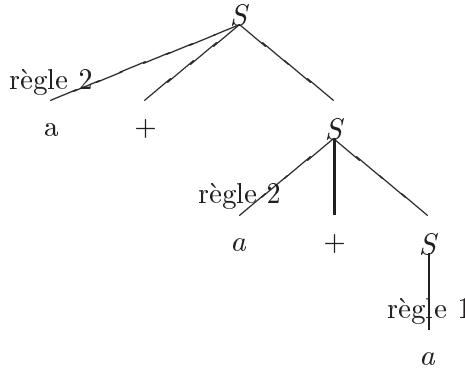


FIG. 7.18 – L'arbre de dérivation de la phrase $a + a + a$ par la grammaire G .

structurée, cette phrase s'écrit : $x = (a + (a + (a)))$

Cette contrainte sur l'échantillon revient à supposer que chaque partie droite de règle de la grammaire qui a engendré cet échantillon est encadrée par une paire de parenthèses, et donc qu'elle appartient à un sous-ensemble des grammaires algébriques ; ce sous-ensemble est strict.

L'algorithme que nous présentons [Sak90] infère des grammaires algébriques *réversibles*, dont la définition formelle est la suivante :

Définition 7.34

Une grammaire algébrique $G = (V, \Sigma, P, S)$ est réversible quand, pour tous non-terminaux A , B et C et toutes séquences α , β et γ de V^* :

- si il y a dans P deux règles $A \rightarrow \alpha$ et $B \rightarrow \alpha$, alors $A = B$, et
- si il y a dans P deux règles $A \rightarrow \alpha B \beta$ et $A \rightarrow \alpha C \beta$, alors $B = C$

Nous donnons cet algorithme sur un exemple. Soit :

$$I_+ = \{((ab)(c)), ((a(ab)b)(c(c))), ((ab)(c(c)))\}$$

La grammaire suivante est d'abord produite : elle ne génère que l'échantillon.

$$\begin{array}{ll} S \rightarrow AB & B \rightarrow c \\ A \rightarrow ab & C \rightarrow aC'b \\ S \rightarrow CD & D \rightarrow cD' \\ C' \rightarrow ab & S \rightarrow EF \\ D' \rightarrow c & F \rightarrow cF' \\ E \rightarrow ab & F' \rightarrow c \end{array}$$

On va procéder à une généralisation par fusion de non-terminaux. Pour préserver la nature réversible de la grammaire inférée, les non-terminaux B , D' et F' sont fusionnés en B , et A , C' et E sont fusionnés en A , ce qui amène la grammaire suivante :

$$\begin{array}{ll} S \rightarrow AB & A \rightarrow ab \\ B \rightarrow c & S \rightarrow CD \\ C \rightarrow aAb & D \rightarrow b \\ S \rightarrow AF & F \rightarrow cB \end{array}$$

De nouveau, il faut fusionner trois non-terminaux : D et F en B , et dans la grammaire résultante, encore fusionner A et C en A . Finalement, on obtient :

$$\begin{aligned} S &\longrightarrow AB & A &\longrightarrow ab \\ A &\longrightarrow aAb & B &\longrightarrow c \\ B &\longrightarrow cB \end{aligned}$$

Cette grammaire reconnaît le langage $\{a^m b^m c^n \mid m, n \geq 1\}$.

7.6.3 Les méthodes par exploration.

Une autre tactique consiste à définir un espace de solutions et à y procéder par cheminement heuristique. C'est tout à fait comparable aux méthodes d'inférence régulière sans échantillon négatif, à ceci près que l'espace de recherche n'est pas défini en pratique de manière aussi structurée que le treillis des automates finis.

Ces méthodes fonctionnent en réduisant implicitement à chaque étape l'ensemble des solutions restantes par modification directe des règles de la grammaire solution courante ; chaque étape correspond à la présentation d'un élément de l'un des ensembles d'apprentissage.

La difficulté est de disposer d'opérateurs de modifications des règles qui soient cohérents avec cette démarche. Une première approche a été proposée par VanLehn et Ball [VB87] : ils ont défini un espace de recherche, montré comment leurs opérateurs permettaient de trouver la grammaire cherchée (si un échantillon suffisant était présenté) et traité les aspects principaux de l'énumération des solutions dans l'optique de l'espace des versions (chapitre 4) ; mais les exemples d'applications qu'ils citent sont très simples, en raison de la nature élémentaire des opérateurs de modification employés. D'autres travaux cherchent à utiliser toute la puissance de la méthode générale de l'espace des versions et mènent à des résultats plus généraux [Gio93].

7.6.4 Une méthode avec oracle.

Cette méthode utilise une propriété des grammaires algébriques : le *lemme de la double étoile*. On peut l'énoncer informellement comme suit :

Propriété 7.1

Pour toute phrase x assez longue d'un langage algébrique L , il existe une décomposition $x = uvwyz$ telle que :

$$\forall k \geq 0, uv^kwy^kz \in L$$

Le principe de la méthode est donc de supposer que l'échantillon possède assez de phrases longues pour aider à découvrir les sous-chaînes v et y , et donc inférer des règles récursives de la forme

$$A \longrightarrow vAz \mid w$$

Pour réduire l'espace de recherche, on suppose avoir à sa disposition un *oracle* (voir le paragraphe 7.2.4), capable de répondre par oui ou non à la question posée par le programme au cours de l'inférence : « Soit la phrase x ; appartient-elle au langage engendré par la grammaire à trouver ? »

À partir de ces deux hypothèses, un algorithme a été présenté par Gonzales et Thomason [GT78]. Sa complexité dépasse de beaucoup celle de son analogue en grammaires régulières³⁵, compte tenu de deux problèmes : d'une part, la combinatoire de décomposition est beaucoup plus

35. Il n'a pas été présenté ici, voir par exemple la référence [Mic76a].

importante (ce qui n'est qu'en partie corrigé par l'usage d'un oracle) ; d'autre part, la découverte d'une récursion n'induit pas directement la grammaire.

Cette méthode est une généralisation et une actualisation des « ancêtres » de l'inférence grammaticale, l'algorithme de Chomsky et Miller [CM57a] pour les langages réguliers et son extension aux langages algébriques par Solomonoff [Sol59]. Elle partage avec elles deux caractéristiques fondamentales : la recherche des récursions comme heuristique de base et l'utilisation d'un oracle.

7.6.5 L'inférence de grammaires linéaires équilibrées

Citons maintenant un algorithme qui concerne une sous-classe des grammaires algébriques.

Définition 7.35

Une grammaire linéaire équilibrée est caractérisée par des règles de la forme : $A \rightarrow uBv \mid w$ avec : $u, v, w \in \Sigma^$, $A, B \in N$ et : $|u| = |v|$;*

Un langage linéaire équilibré est tel qu'il existe une grammaire linéaire équilibrée qui l'engendre.

La famille de ces langages algébriques est strictement supérieure à celle des langages réguliers.

Takada [Tak88] utilise la notion suivante :

Définition 7.36

Un langage est dit engendré par une grammaire $G = (V, \Sigma, S, P)$ sous le contrôle d'un sous-ensemble $C \in P^$ quand toutes ses phrases se dérivent de S par une suite de règles de P formant un élément de C .*

Le sous-ensemble C peut être vu comme un langage sur P^* : il peut donc être régulier, algébrique, etc.

La méthode construit d'abord une certaine grammaire G_L , linéairement équilibrée universelle (autrement dit telle que $L(G_L) = X^*$), et utilise la propriété suivante :

Propriété 7.2

Pour tout langage linéaire équilibré L , il existe un unique sous-ensemble régulier $C \in P^$ tel que L soit engendré par G_L sous le contrôle de C .*

Le problème de l'inférence d'une grammaire linéairement équilibrée peut alors se ramener à celui de la construction d'un ensemble de contrôle éventuellement infini, mais régulier ; Takada propose un algorithme général d'inférence de ces grammaires dont la partie inductive peut être composée à partir de n'importe quel algorithme d'inférence régulière.

7.7 Quelques extensions

7.7.1 Les grammaires stochastiques

Nous avons traité jusqu'ici des algorithmes d'apprentissage de concept sous la forme d'une grammaire. Pour toute phrase sur Σ^* , on sait décider si elle appartient ou non au langage engendré par une grammaire régulière ou algébrique. L'espace de représentation est donc partagé en deux par un concept grammaire. Une extension de l'inférence grammaticale consiste à étudier l'apprentissage de *grammaires stochastiques*.

Une grammaire stochastique est constituée comme une grammaire ordinaire, mais à chaque règle est associée une probabilité d'application, chaque non-terminal ayant une probabilité totale de 1 d'être appliquée. Par exemple, la grammaire suivante est régulière stochastique :

$$\begin{array}{lcl} S & \xrightarrow{0.8} & aA \\ S & \xrightarrow{0.2} & a \\ A & \xrightarrow{0.6} & bA \\ A & \xrightarrow{0.4} & b \end{array}$$

Elle correspond à l'*automate fini stochastique* de la figure 7.19

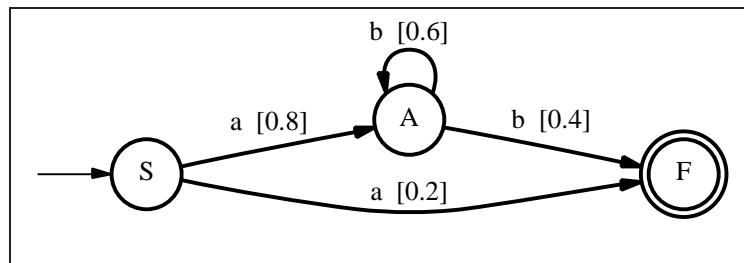


FIG. 7.19 – Un automate fini stochastique.

La probabilité d'une phrase par rapport à une grammaire se calcule comme la somme, sur toutes les façons possibles que cette phrase a d'être émise par la grammaire, du produit des probabilités des règles utilisées. Par exemple, la phrase *abb* ne peut être acceptée par l'automate de la figure 7.19 que par la succession de transitions $S \xrightarrow{0.8} A \xrightarrow{0.6} A \xrightarrow{0.6} A \xrightarrow{0.4} F$: il n'y a qu'une seule façon de l'émettre. Sa probabilité est : $0.8 \times 0.6 \times 0.6 \times 0.6 \times 0.4 = 0.0069$. On la notera ici : $abb[0.0069]$. De même, la phrase *bb*, non acceptée par l'automate, a une probabilité nulle : $bb[0]$.

Sous certaines conditions, on peut assurer qu'une grammaire stochastique est *consistante*³⁶, c'est-à-dire qu'elle définit une distribution de probabilités sur Σ^* . Toujours dans notre exemple, le langage engendré peut s'écrire :

$$\{a[0.2]\} \cup \{ab^n[0.8 \times 0.6^{n-1} \times 0.4] \mid n \geq 1\}$$

On peut vérifier que la grammaire proposée est consistante.

Un *échantillon d'inférence stochastique* est constitué d'un ensemble de phrases muni de fréquences d'apparition. On peut donc aussi le considérer comme un *multiensemble* de séquences. Par exemple, le multiensemble :

$$\{a, ab, ab, ab, abb, abb, abbbb\}$$

peut se récrire :

$$\{a[\frac{1}{7}], ab[\frac{3}{7}], abb[\frac{2}{7}], abbbb[\frac{1}{7}]\}$$

Pour en rester au cas régulier, l'inférence consiste donc à apprendre un automate fini stochastique à partir d'un tel échantillon d'inférence.

36. L'emploi de ce mot est ici sans ambiguïté.

Nous ne rentrerons pas ici dans le détail des méthodes. Donnons le principe commun à plusieurs d'entre elles ([CO94], [Car97], [JP98], [TD]), qui est une généralisation de la technique RPNI (sans exemples négatifs) expliquée au paragraphe 7.5.4.

- On construit le *PTA* stochastique, qui génère au maximum de vraisemblance l'échantillon d'apprentissage stochastique.
- Le processus de généralisation est la fusion d'états.
- Quand deux états sont fusionnés, il faut recalculer les probabilités associées à ses transitions de sortie de manière à garantir la consistance du langage accepté.
- Le choix des deux états à fusionner se fait par exemple en cherchant à maximiser la ressemblance entre la distribution de probabilités sur l'échantillon et celle donnée sur Σ^* par l'automate obtenu ou sur un test statistique.
- Comme il n'y a pas de contre-exemple, l'arrêt du processus doit se faire par un critère empirique, qui établit un compromis entre la complexité de l'automate inféré et son adéquation avec l'échantillon.

Notons qu'il existe une extension stochastique de l'algorithme ECGI qui permet d'inférer une grammaire régulière stochastique par une procédure incrémentale en une seule passe sur l'échantillon positif [CM99]. En effet, les probabilités associées aux productions de la grammaire peuvent être mises à jour par comptage de la fréquence d'utilisation des règles lors de l'analyse corrective de chaque nouvelle chaîne.

7.7.2 Le point de vue connexionniste

Terminons cette section en ouvrant un aperçu sur un certain nombre de travaux qui portent sur l'utilisation de *réseaux connexionnistes* en inférence grammaticale, habituellement à partir d'échantillons positif et négatif. Le principe de cet apprentissage ne correspond pas au cadre classique de l'inférence tel que nous l'avons présenté. Cependant une équivalence théorique entre certains réseaux connexionnistes et les automates finis déterministes a été démontrée.

L'idée est d'utiliser l'apprentissage par réseaux connexionnistes (voir le chapitre 10) ayant pour entrées des séquences et non plus des vecteurs de \mathbb{R}^d . Il faut pour cela utiliser des architectures plus complexes que celles des réseaux multicouches. Parmi les nombreuses variantes, les réseaux connexionnistes récurrents, et les réseaux d'ordre 2 ont été utilisés pour approcher le comportement d'automates finis. Des extensions de ces modèles ont été proposées afin d'approximer les *automates à piles* qui sont équivalents aux grammaires algébriques. Une bonne référence récente sur ces sujets est [SG98].

Notes historiques et sources bibliographiques

L'inférence grammaticale est désormais à la convergence de plusieurs domaines qui se sont développés plus ou moins indépendamment :

- l'algorithme d'apprentissage pour la *reconnaissance structurelle des formes* [Fu74] ;
- la théorie de l'apprentissage [Pit89] ;
- les sciences cognitives .

Les algorithmes actuels sont très puissants. *RPNI* et ses dérivés sont capables d'apprendre des automates de plusieurs centaines de milliers d'états à partir d'échantillons de très grande taille. Ceci permet de représenter des concepts aussi complexes que des modèles de la langue écrite. Le développement des algorithmes d'inférence stochastiques a parallèlement permis l'apprentissage

de concepts qui ressemblent à celui dont l'échantillon provient. Les applications sont désormais d'une grande variété et d'une grande efficacité. Citons en vrac comme domaines d'application :

- les bioséquences ([AM97], [BC97]) ;
- la structure des textes électroniques ([AMN94], refyoun99) ;
- le traitement du langage naturel ([Vid94a]) ;
- la traduction automatique ([CGV94]) ;
- la reconnaissance de la parole ([CM99]) ;
- l'étude des styles musicaux ([CAVR98]) ;
- la reconnaissance des caractères manuscrits ([LVA⁺94]) ;
- la compression de données ([NMWM94]).

La théorie de l'inférence grammaticale régulière est particulièrement bien décrite dans [Pit89]. L'analyse algébrique de l'espace de recherche a été décrite dans [DMV94]. Les performances des algorithmes sont détaillés dans [LPP98]. Les deux articles [Sak97] et [Lee96] décrivent particulièrement l'inférence de grammaires algébriques, tandis que [TD99] est assez complet sur l'inférence stochastique. Une partie du texte de ce chapitre est repris de [DM98] et de [Dup96].

On trouvera dans les articles bibliographiques des références à des techniques du même type qui permettent l'apprentissage de transducteurs (des automates finis qui transforment des séquences en d'autres séquences), à des grammaires d'arbres, à des grammaires de formes bidimensionnelles, etc.

Le site internet : <http://www.univ-st-etienne.fr/eurise/gi/gi.html> est très complet sur l'inférence grammaticale. Il renvoie en particulier à des bibliographies thématiques et à des textes de présentation.

On doit aussi citer, dans un registre voisin, les travaux de D. Osherson, P. Adriaans, etc. ([OSW86]) qui s'intéressent à l'apprentissage de fonctions récursives.

Résumé

- L'inférence grammaticale apprend un concept à partir d'exemples positifs et négatifs de séquences.
- En général, ces concepts sont des automates finis : ce sont des modèles simples, mais très utiles.
- La théorie et les algorithmes de l'inférence grammaticale sont des thèmes riches et actifs en apprentissage.
- Des extensions existent pour les automates stochastiques, certains types de transducteurs et des modèles grammaticaux plus complexes, comme les grammaires d'arbres.

Chapitre 8

Apprentissage par évolution simulée

La recherche d'une bonne hypothèse est généralement très difficile car les espaces d'hypothèses sont de taille considérable, souvent infinie, et que la fonction d'évaluation y est généralement irrégulière. C'est pourquoi il est avantageux de disposer soit d'une structure forte sur \mathcal{H} , comme c'est le cas lorsque l'on peut utiliser une relation de généralité entre hypothèses, soit de connaissances a priori sur le domaine permettant de contraindre la recherche. À défaut de quoi, on est condamné à recourir à des méthodes d'exploration plus faibles, telle les méthodes de gradient plus ou moins sophistiquées (comme le recuit simulé).

Les algorithmes évolutionnaires occupent une place intermédiaire. Le principe est de faire une recherche dans l'espace \mathcal{H} en utilisant une population d'hypothèses que l'on fait évoluer selon les recettes de l'évolution naturelle : en utilisant des opérateurs génétiques de croisement et de mutation, et en sélectionnant les meilleurs individus pour engendrer la génération suivante d'hypothèses. D'un côté, on peut considérer qu'il s'agit là d'une sorte de méthode de gradient stochastique utilisant une population plutôt qu'une seule hypothèse. De l'autre, on peut aussi voir ces algorithmes comme des méthodes d'exploration guidées car elles découpent l'espace d'hypothèses en l'espace d'hypothèses proprement dit et un espace « génotypique » dans lequel des opérateurs d'évolution guident la recherche de meilleures hypothèses.

C'est en raison de ces deux visages que nous avons placé l'exposé de ces méthodes à la charnière de la partie sur l'apprentissage par exploration (guidée) et de la partie sur l'apprentissage par optimisation (utilisant des méthodes de gradient). Cette dualité les rend très attrayantes et importantes à connaître. Elles sont d'un large emploi dans les applications.

LES CANARDS ne sont pas tous sauvages. Certains sont domestiqués et modifiés à des fins alimentaires ou esthétiques. Pour ce faire, les éleveurs « créent » des races par des manipulations génétiques naturelles, essentiellement par des croisements visant à améliorer tel ou tel critère sur l'oiseau : sa qualité gustative, la quantité de sa ponte ou certains aspects de son apparence. Voici deux exemples de ce que l'on peut lire dans une encyclopédie récente des animaux domestiques¹ :

Canard Kaki Campbell. Cette race doit son nom à une éleveuse anglaise du Gloucestershire, Mrs Campbell, qui la créa en 1901 à partir de canard sauvage, de Coureur indien brun et blanc et peut-être d'Orpington. On attribue à une éleveuse, Mme Flamencourt, les premières sélections françaises dans les années 1920. (...) La cane est utilisée en croisement avec des mâles Pékin pour la production de canards dits Nantais.

Canard Huppé. Mutation du canard fermier. Les sujets les mieux huppés ont été sélectionnés par des éleveurs séduits par cette singularité afin de créer un standard.

On voit donc que l'homme peut utiliser avec succès les principes darwiniens de la sélection pour optimiser sa production domestique. Il est moins certain que l'on puisse considérer l'évolution naturelle comme une optimisation, non pas parce que les espèces existantes sont ratées, mais parce que le critère que la Nature chercherait à optimiser est difficile à formuler et peut-être inexistant. Restons donc sur l'idée que, étant donné un animal cible, il est parfois possible de fabriquer un animal proche de cette cible, en utilisant des croisements et en profitant des mutations naturelles.

En quoi cette constatation peut-elle être utile à l'apprentissage ? D'abord, comme on l'a vu, l'apprentissage artificiel est étroitement relié à l'optimisation, soit sous la forme de l'exploration d'un espace structuré, soit par l'utilisation de propriétés analytiques. Ensuite, puisqu'il est possible de modifier avec une grande rapidité (quelques dizaines de générations) le code génétique d'animaux complexes dans une direction choisie, on peut se demander si une modélisation informatique de ce code et la programmation d'une technique singeant l'évolution naturelle ne seraient pas une bonne technique d'optimisation.

Cette question a été posée dès les années 1960 et la réponse a été positive sous plusieurs angles. D'abord, les algorithmes dits génétiques, fondés sur l'idée précédente, ont connu un grand succès dans l'optimisation de fonctions compliquées et peu régulières, au prix cependant d'un temps important de calcul. Plus intéressant pour l'apprentissage, de plus en plus indépendamment de la métaphore biologique, des techniques d'évolution de code, au sens informatique cette fois, ont été développées. C'est ce que l'on appelle la programmation génétique. Le but de ce chapitre est de présenter un point de vue unifié sur ces techniques et d'en montrer la puissance et l'intérêt pour l'apprentissage artificiel.

1. A. Raveneau: *Inventaire des animaux domestiques en France (bestiaux, volailles, animaux familiers et de rapport)*. Nathan, 1993.

Notations utiles pour ce chapitre

p_{mut}	Probabilité de mutation d'un bit
p_{crois}	Probabilité de croisement
$P_{\mathcal{H}}(t)$	Population d'hypothèses dans \mathcal{H} à l'instant t .
$P_{\mathcal{G}}(t)$	Population de génomes dans \mathcal{G} à l'instant t .
N	Taille de la population $P_{\mathcal{G}}$
ϕ	La fonction de performance (fitness function)
t	Le nombre d'individus participant à un tournoi
L	La longueur des chaînes de bits des génomes de \mathcal{G}
$n(s, t)$	Nombre de génomes représentants le schéma s dans la population $P_{\mathcal{G}}(t)$

8.1 Trois espaces au lieu de deux

Jusqu'à présent, à l'exception du chapitre 6, l'apprentissage a été abordé comme un jeu entre deux espaces :

- l'*espace des exemples* \mathcal{X} , correspondant au monde à décrire et à interpréter. L'objectif de l'apprentissage se traduit souvent par une fonction de risque permettant d'évaluer la qualité de la description apprise du monde ;
- l'*espace des hypothèses* \mathcal{H} , correspondant à l'espace des descriptions du monde ou encore des modèles ou théories.

Le but de l'apprentissage est de trouver une, ou plusieurs, bonne(s) description(s) du monde \mathcal{X} dans \mathcal{H} . De ce fait, l'espace des hypothèses est chargé de trois missions :

1. Il doit *réaliser un biais adéquat*, c'est-à-dire pouvoir décrire les modèles du monde performants dans l'application visée, tout en étant suffisamment limité dans son expressivité pour que l'induction soit possible et peu coûteuse en terme de taille d'échantillon de données. Ce prérequis a été largement analysé dans le chapitre 2.
2. Il doit aussi correspondre à un *langage d'expression des hypothèses* $\mathcal{L}_{\mathcal{H}}$ aussi intelligible que possible s'il est désirable qu'un expert humain puisse interpréter ce qui est appris, y compris pour orienter le système apprenant. C'est pourquoi, quand c'est possible, on préférera souvent un apprentissage de règles plutôt qu'un apprentissage par réseau connexionniste, dont le résultat est plus difficile à interpréter.
3. Finalement, il doit *posséder une topologie favorisant une exploration efficace*, c'est-à-dire telle qu'elle permette d'atteindre une hypothèse ou une combinaison d'hypothèses quasi optimale avec un coût d'exploration limité. En général, l'exploration se fait par modification des hypothèses courantes par l'emploi d'opérateurs syntaxiques simples opérant sur l'expression des hypothèses dans le langage $\mathcal{L}_{\mathcal{H}}$. Le problème est de trouver des opérateurs permettant d'atteindre n'importe quel point de \mathcal{H} à partir de n'importe quel autre point (propriété de complétude) et conduisant naturellement aux régions de \mathcal{H} les plus prometteuses.

Ces trois missions ne sont pas forcément faciles à remplir simultanément. Il n'est généralement pas facile de trouver un langage $\mathcal{L}_{\mathcal{H}}$ à la fois limité du point de vue du biais inductif, mais expressif et intelligible, tout en possédant des propriétés syntaxiques favorisant l'exploration. Pourquoi alors ne pas découpler ces objectifs ? C'est ce que réalisent les algorithmes d'évolution simulée.

L'idée est la suivante : à l'espace des hypothèses \mathcal{H} est associé un autre espace appelé, pour des raisons de similarité avec les mécanismes de l'évolution naturelle, espace « génotypique »

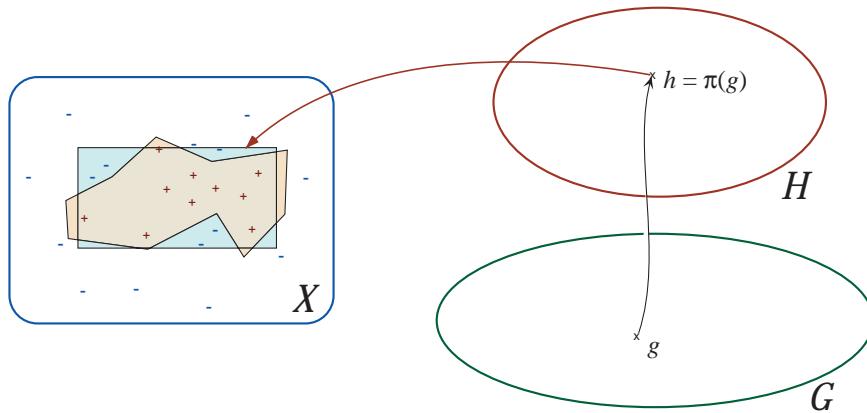


FIG. 8.1 – L’induction est ici réalisée par un jeu entre trois espaces : d’abord l’espace des exemples \mathcal{X} , ensuite l’espace des hypothèses \mathcal{H} dans lequel est cherchée une description adéquate de \mathcal{X} , et enfin l’espace « génotypique » \mathcal{G} dans lequel s’effectuent les opérations permettant l’exploration de \mathcal{H} .

et noté \mathcal{G} . Lorsqu’il faut changer d’hypothèse h_t dans \mathcal{H} , on prend l’expression génotypique associée g_t dans \mathcal{G} à laquelle on fait subir des opérateurs de modification pour obtenir de nouvelles expressions g_{t+1} correspondant à de nouvelles hypothèses h_{t+1} qui sont alors testées pour mesurer leur performance dans le monde \mathcal{X} . Ce cycle d’évaluation des hypothèses dans \mathcal{H} et de modification via \mathcal{G} se déroule jusqu’à ce qu’une bonne hypothèse soit trouvée ou qu’il soit décidé de consacrer les ressources computationnelles à d’autres tâches, y compris éventuellement à une autre passe d’apprentissage en partant de conditions initiales différentes (voir la figure 8.1).

Le découplage des missions dévolues à l’espace des hypothèses avec un espace spécial dédié à la mission d’exploration est une idée intéressante. Le problème est d’identifier les espaces \mathcal{G} appropriés. Plusieurs écoles se sont développées en concurrence depuis les années 1960, chacune affirmant avoir identifié le bon espace \mathcal{G} avec ses opérateurs associés. Se distinguent :

- Les *algorithmes génétiques* développés par John Holland à partir des années 1960 et analysés en profondeur en 1975 [Hol75]. Holland a ainsi proposé une analyse théorique, le « théorème des schémas », montrant selon lui la supériorité d’un espace génotypique décrit dans un langage de chaînes de bits. Les références à la métaphore de l’évolution naturelle sont les plus explicites dans ce travail et le principe général de ces algorithmes a servi de standard pour les autres approches.
- Les *stratégies d’évolution* développées par Rechenberg et Schwefel en 1965 pour la résolution de problèmes d’optimisation de structures mécaniques. Cette école préconise l’utilisation de \mathbb{R} pour l’espace génotypique.
- La *programmation évolutive*, développée par Fogel à partir de 1966, étudie l’exploration d’un espace d’automates à états finis (FSA) pour décrire certains types de fonctions et de programmes.
- La *programmation génétique*, défendue avec force par Koza depuis 1992, promeut l’usage d’un espace d’arbres décrivant des programmes dans un langage adéquat (par exemple des expressions du langage Lisp).

Si les applications et les idées de développement sont foisonnantes pour l’ensemble de ces approches, les arguments théoriques expliquant leur fonctionnement et démontrant la supériorité de l’une d’entre elles sur les autres sont encore au stade exploratoire. Beaucoup reste donc à faire dans ce domaine.

Après une description générique de ces approches, ce chapitre étudie brièvement tour à tour les quatres familles de méthodes décrites ci-dessus. Une idée intéressante qui s'impose naturellement avec ces approches d'*évolution simulée* est celle de populations d'hypothèses se développant en symbiose. Ce phénomène est appelé « coévolution ». Ce chapitre se termine par sa description et par les perspectives ouvertes vers le développement d'« écologies » d'hypothèses, c'est-à-dire des théories (au sens de fragments de connaissance organisés), plutôt que sur la recherche d'une hypothèse unique.

Afin de souligner l'idée essentielle, selon nous, du passage par un espace dédié à la mission d'exploration, nous appellerons ces familles de techniques « algorithmes opérant sur le génotype » ou encore AG. Nous devons avertir le lecteur que cela est original et qu'en général on parle plutôt d'algorithmes d'évolution simulée (ES), tandis que l'acronyme AG est réservé à l'approche des algorithmes génétiques.

8.2 Un modèle formel simplifié de l'évolution

Cette section introduit les concepts fondamentaux communs aux algorithmes opérants sur le génotype ainsi que l'algorithme d'apprentissage général.

8.2.1 Le jeu entre \mathcal{H} et \mathcal{G}

Nous considérons désormais que l'espace des hypothèses \mathcal{H} est couplé à un espace génotypique \mathcal{G} . Quelle est la nature de ce couplage? Notons π la relation liant \mathcal{G} à \mathcal{H} . En général, il ne s'agit pas d'une bijection qui associe à chaque h un unique g . S'il y a en effet toujours surjection : chaque hypothèse h est l'image par π d'au moins un élément $g \in \mathcal{G}$ (\mathcal{H} étant le plus souvent défini par référence à \mathcal{G} : $\mathcal{H} = \pi(\mathcal{G})$), il n'y a pas nécessairement injection : plusieurs éléments g pouvant être projetés sur une même hypothèse h . On parlera donc à raison de dualité entre \mathcal{G} et \mathcal{H} , mais ce sera par abus de langage que l'on parlera de couple (g, h) .

Dans cette dualité $(\mathcal{G}, \mathcal{H})$, la répartition des rôles est la suivante :

- La *mesure de performance* s'effectue dans \mathcal{H} : on mesure la valeur d'une hypothèse donnée en la confrontant, par l'intermédiaire d'une fonction de risque, appelée ici mesure de performance (*fitness function*), aux données d'apprentissage fournies dans \mathcal{X} . On notera η cette fonction de performance: $\eta: \mathcal{H} \times \mathcal{X} \rightarrow \mathbb{R}$.
- La *variation* concommittante à l'apprentissage s'effectue dans \mathcal{G} . En réponse à la mesure de performance attachée à $\pi(g_t) = h_t$, le système calcule un ou plusieurs nouveaux éléments g_{t+1} correspondant à une ou plusieurs nouvelles hypothèse(s).

8.2.2 L'apprentissage comme processus d'évolution d'une population

Ce processus d'apprentissage par jeu entre un espace \mathcal{H} d'individus se mesurant à l'environnement et un espace \mathcal{G} responsable du calcul de nouveaux individus a été en grande partie influencé par la métaphore de l'évolution des espèces telle que décrite par Darwin en 1859 [Dar59]. C'est pourquoi on qualifie souvent l'espace \mathcal{H} de phénotypique: celui où s'expriment les traits des individus dont le génotype est décrit dans \mathcal{G} , l'espace génotypique. Par ailleurs, les méthodes d'apprentissage issues de cette métaphore utilisent souvent une population d'individus pour réaliser l'apprentissage plutôt qu'un individu unique. Dans ce cas, on considère une population $P_{\mathcal{G}}(t)$ d'éléments de \mathcal{G} à l'instant t qui, en fonction des performances des hypothèses de la population $P_{\mathcal{H}}(t)$ associée, détermine une nouvelle population $P_{\mathcal{G}}(t+1)$ à l'étape suivante. On a alors le processus schématisé dans la figure 8.2 et dont les étapes du cycle général sont les suivantes :

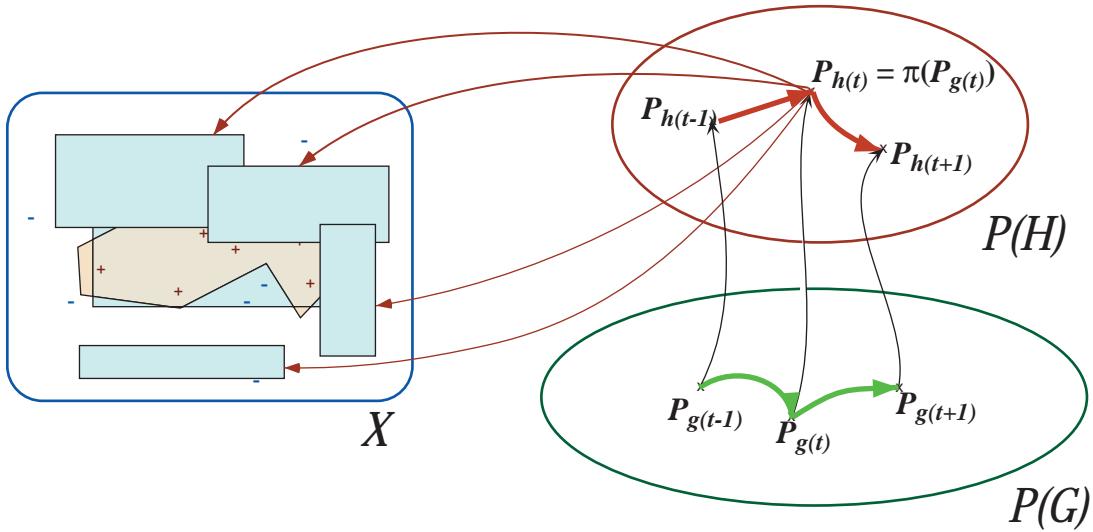


FIG. 8.2 – Le processus général d'apprentissage par l'utilisation d'un espace génotypique. Nous avons noté ici $\mathcal{P}(\mathcal{H})$ et $\mathcal{P}(\mathcal{G})$ des espaces de populations d'hypothèses et de génomes. De ce fait, dans ce schéma, il faut comprendre $P_{\mathcal{H}(t)}$ et $P_{\mathcal{G}(t)}$ respectivement comme une population d'hypothèses à l'instant t et une population de génomes à l'instant t . Ces populations évoluent en cours de recherche.

Étape 1 **Expression**: une population $P_{\mathcal{G}}(t) \in \mathcal{G}$ est projetée dans l'espace phénotypique \mathcal{H} par l'application π : $P_{\mathcal{H}}(t) = \pi(P_{\mathcal{G}}(t))$.

Étape 2 **Mesure de la performance**: la population d'hypothèses est évaluée, en général par une évaluation individuelle de chaque hypothèse, grâce à la fonction ϕ qui utilise les données d'apprentissage. Cela évalue implicitement la population $P_{\mathcal{G}}(t)$.

Étape 3 **Sélection d'une opération de variation de $P_{\mathcal{G}}(t)$** : en fonction de l'évaluation de la population $P_{\mathcal{H}}(t)$, le système sélectionne l'opération de variation qui va transformer la population $P_{\mathcal{G}}(t)$.

Étape 4 **Variation**: l'opération de variation sélectionnée à l'étape précédente est appliquée à $P_{\mathcal{G}}(t)$ pour obtenir la nouvelle population $P_{\mathcal{G}}(t + 1)$.

Ce cycle est répété jusqu'à ce qu'une population $P_{\mathcal{H}}(t)$ obtienne une évaluation jugée satisfaisante (par exemple la meilleure hypothèse $h \in P_{\mathcal{H}}(t)$ ne change plus ou très peu du point de vue de sa performance), ou jusqu'à ce qu'une borne de ressources computationnelles soit atteinte.

Nous allons illustrer ce processus général en étudiant le cas des algorithmes génétiques.

8.3 Les algorithmes génétiques

Les algorithmes génétiques ont été popularisés par John Holland en 1975 grâce à son livre *Adaptation in natural and artificial systems* (réédité en 1992) [Hol75]. Dans ce travail, Holland cherche à utiliser la métaphore de l'adaptation naturelle des espèces par variation et sélection dans le but de réaliser automatiquement une adaptation optimale à l'environnement. À cet effet, Holland met en avant quatre principes:

1. la *dualité* entre un espace génotypique chargé des opérations de variation d'une génération à la suivante et un espace phénotypique dans lequel se produit la sélection;

2. un *codage* des éléments de \mathcal{G} par un alphabet discret, à l'instar de l'alphabet à quatre lettres de l'ADN (Holland milite pour un alphabet binaire pour des raisons théoriques qui seront étudiées à la section 8.3.6) ;
3. une variation entre générations assurée par des *opérateurs génétiques*, en particulier *sexués* ;
4. le concept de *primitives (building blocks)* dans le code génétique qui se combinent pour assurer des interactions complexes nécessaires à la construction d'hypothèses sophistiquées.

Nous avons décrit l'idée fondamentale de dualité dans l'introduction de ce chapitre. Tournons-nous maintenant vers le problème du codage utilisant un alphabet binaire.

8.3.1 La représentation dans l'espace génotypique

Les algorithmes génétiques standards (car il y a des mutants, évidemment) utilisent une représentation dans l'espace des génotypes \mathcal{G} , ou encore des génomes, par des chaînes de bits de longueur fixe. Par exemple, chaque hypothèse de \mathcal{H} pourrait être associée à un génome de longueur 12 bits, permettant ainsi de représenter 2^{12} individus. Du point de vue pratique, l'intérêt de ce type de représentation est qu'il se prête à des opérateurs de modification simples (voir section 8.3.4). Le théorème des schémas (voir section 8.3.6) revendique une justification théorique de l'emploi de représentations binaires.

La question est de savoir comment choisir la représentation adéquate pour le problème étudié. Cela implique un choix délicat de la taille de la représentation : plus celle-ci est grande, plus le nombre d'hypothèses représentables est élevé, mais plus aussi l'exploration de \mathcal{G} et donc de \mathcal{H} peut être coûteuse. Généralement, le choix du codage par des bits est lié à celui des opérateurs de modification des génomes, dans la mesure où il faut éviter qu'un opérateur puisse, en « cassant » les primitives du codage, engendrer des génomes ne correspondant à aucune hypothèse valide.

8.3.2 L'algorithme générique

L'algorithme génétique standard règle l'évolution d'une population de génomes sous la pression sélective des performances des hypothèses correspondantes dans \mathcal{H} . Il est dans ses grandes lignes décrit par les étapes de l'algorithme 8.1.

Algorithme 8.1 Algorithme génétique standard.

1. **Initialisation** de la population $P_{\mathcal{G}}(0)$.
 2. **Évaluation** de la population courante $P_{\mathcal{G}}(t)$ par un calcul de la performance de chaque hypothèse h associée à chaque génome $g \in P_{\mathcal{G}}(t)$.
 3. **Sélection** des μ meilleurs génomes de $P_{\mathcal{G}}(t)$ qui vont devenir les « procréateurs » de la génération suivante $P_{\mathcal{G}}(t + 1)$.
 4. **Modification** des procréateurs par des opérateurs génétiques et obtention de λ « descendants ».
 5. **Remplacement** de la population courante par les individus résultant des opérations génétiques de l'étape 4.
 6. Tant que le critère d'arrêt n'est pas satisfait, retour en 2.
-

Nous allons aborder ces différentes étapes une par une.

8.3.3 L'initialisation de la population

La plupart des algorithmes opérant sur le génotype organisent l'exploration de l'espace \mathcal{G} (donc indirectement de \mathcal{H}) en s'appuyant sur une population d'individus qui évoluent en essayant de maximiser la performance mesurée dans \mathcal{H} . Deux questions découlent de cette technique : d'abord, quelle est la *taille optimale de la population* pour favoriser l'exploration efficace de \mathcal{G} , ensuite, *comment initialiser* les individus de la première génération $P_{\mathcal{G}}(0)$?

Aucune de ces deux questions n'a de réponse théorique bien établie, et le choix de ces deux paramètres est donc souvent le résultat de jugements plus ou moins éclairés sur la nature du problème étudié.

La taille de la population de génomes, généralement fixe au cours du déroulement de l'algorithme et notée N , est fréquemment de l'ordre de plusieurs centaines d'individus dans les algorithmes génétiques. En effet, les justifications théoriques de ces algorithmes indiquent qu'il faut des populations très grandes (idéalement infinies) pour que l'algorithme fonctionne optimalement. Cela est aussi souhaitable pour conserver une bonne diversité à la population, permettant ainsi d'échapper aux optima locaux. Inversement, les considérations pratiques de coût computationnel militent pour l'emploi de populations de taille limitée, sous peine de ne pas pouvoir répéter suffisamment les étapes nécessaires à l'évolution de la population.

Par ailleurs, le choix de la population initiale s'effectue souvent par un tirage aléatoire sur chaque bit des chaînes constituant les individus de \mathcal{G} . Cependant, si des connaissances *a priori* sont disponibles sur le problème, il est possible de biaiser la population initiale en sélectionnant des individus représentant des hypothèses prometteuses. Il faut cependant se méfier de ne pas pour autant trop amoindrir la diversité génétique nécessaire à l'exploration par évolution simulée.

8.3.4 Les opérateurs

Dans le cadre des algorithmes opérant sur le génotype, on appelle *opérateurs* des applications définies sur \mathcal{G} permettant le renouvellement de la population à chaque génération. La variété des opérateurs imaginables est infinie. C'est à la fois un des charmes des travaux exploratoires dans ce domaine, mais aussi une source d'agacement car bien souvent seul le concepteur d'un nouvel opérateur en voit clairement l'intérêt ! John Holland a proposé de suivre les recettes de l'évolution naturelle en adoptant les deux principaux opérateurs connus vers le milieu du XX^e siècle² : la reproduction sexuée impliquant un *croisement* entre génomes, et la *mutation* par erreur de recopie ou par altération due à un agent extérieur (par exemple des rayons cosmiques).

8.3.4.1 Les opérateurs de croisement

Un opérateur de croisement est une application de: $\mathcal{G} \times \mathcal{G} \rightarrow \mathcal{G} \times \mathcal{G}$. Il prend donc deux génomes et produit deux nouveaux génomes. De manière analogue au croisement (*cross-over*) de la génétique naturelle, l'idée essentielle est de combiner des caractéristiques des parents dans l'espoir d'obtenir des descendants bénéficiant du meilleur des deux parents. Dans le cas d'une représentation par chaîne de bits de longueur fixe, le croisement consiste à sélectionner des sous-chaînes de chaque génome et à les intervertir. On distingue le croisement à un point et le croisement multipoint.

– Dans le **croisement à un point**, on tire d'abord au hasard un point entre deux bits de

2. Depuis, les recherches sur la génétique ont dévoilé d'autres opérateurs qui semblent guider l'évolution et permettre par exemple une adaptation plus rapide quand des changements de milieu qui ont déjà été rencontrés dans le passé surviennent (voir par exemple le livre de Christopher Wills *La sagesse des gênes*) et le livre de science fiction *L'échelle de Darwin (Darwin's radio)* de Greg Bear pour un roman basé sur ces idées.

la représentation par chaîne, puis en prenant les deux génomes parents, on échange leurs sous-chaînes à partir de ce point. (Voir figure 8.3 (a)).

- Dans le **croisement multipoint**, on tire d'abord au hasard plusieurs points entre bits de la représentation par chaîne, puis en prenant les deux génomes parents, on échange les sous-chaînes correspondantes. (Voir figure 8.3 (b)).
- Dans le cas extrême, les sous-chaînes échangées sont réduites à un bit. Chaque bit a alors une certaine probabilité, souvent proche de 0.5, d'être échangé avec le bit correspondant de l'autre génome. On parle alors de **croisement uniforme**. (Voir figure 8.3 (c)).

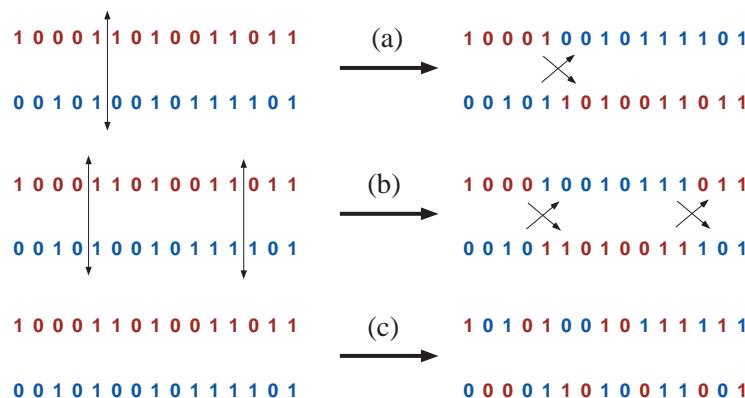


FIG. 8.3 – En (a) un exemple de croisement à un point. En (b) un exemple de croisement multipoint. En (c) un exemple de croisement uniforme.

8.3.4.2 Les opérateurs de mutation

Un opérateur de mutation est une application de: $\mathcal{G} \rightarrow \mathcal{G}$. Il prend en entrée un génome (une chaîne de bits) et retourne une autre chaîne de bits. On parle de « mutation » pour indiquer qu'à l'instar des mutations génétiques rencontrées dans la nature, la transformation n'affecte que quelques bits de la chaîne. Un opérateur de mutation fonctionne en changeant aléatoirement, suivant une probabilité p_{mut} , chaque bit de la chaîne passée en argument dont la valeur est inversée ((Voir figure 8.4). En général la probabilité p_{mut} est faible, juste suffisante pour assurer la mutation de quelques génomes dans la population $P_{\mathcal{G}}(t)$. Mais la proportion idéale de mutations est sujette à débats.

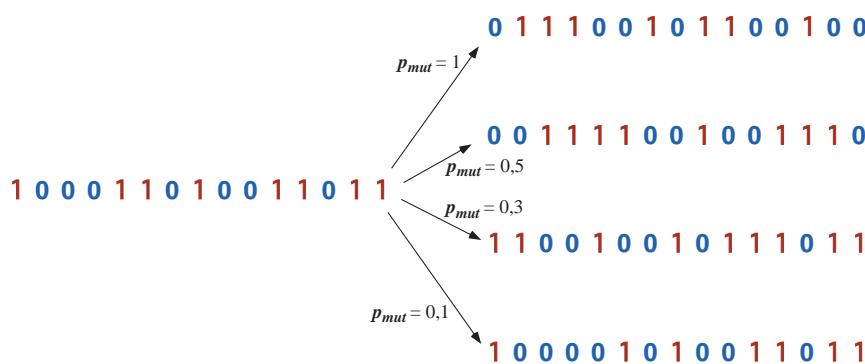


FIG. 8.4 – Un exemple d'opération de mutation sur une chaîne de bits.

8.3.4.3 Débat : le rôle des opérateurs

Une manière d'envisager les algorithmes génétiques est de les considérer comme des algorithmes d'optimisation par recherche dans un espace de possibilités. De tels algorithmes font face au dilemme classique « exploitation contre exploration ». L'*exploitation* utilise l'information acquise, par exemple sous la forme d'un gradient local, pour modifier l'estimation courante. Ce genre de stratégie conduit le plus souvent à des optima locaux. L'*exploration* teste des solutions ailleurs que dans le voisinage immédiat de la solution courante, parfois par tirage aléatoire. Cette seconde stratégie permet d'explorer plus largement l'espace de recherche et d'échapper ainsi aux optima locaux. En revanche, en n'exploitant pas l'information acquise, elle est entièrement soumise aux aléas du tirage et ressemble à une marche aléatoire. Certaines méthodes de recherche, comme le *recuit simulé*, implémentent un compromis entre l'exploitation et l'exploration (voir aussi certaines méthodes d'apprentissage par renforcement (chapitre 16)). C'est également le cas des algorithmes génétiques par l'usage des opérateurs de croisement et de mutation.

La recherche en génétique laisse supposer que l'échange de matériel génétique entre individus joue un grand rôle dans l'évolution des organismes avancés. Le principe en serait que ce *croisement* permet une recombinaison de primitives déjà testées (blocs d'ADN ou séquences de bits) ouvrant la possibilité de combinaisons encore plus performantes entre ces primitives. Par combinaisons successives de blocs de plus en plus grands, le croisement permettrait ainsi l'évolution vers des organismes de plus en plus performants tendant à être retenus par le processus de sélection naturelle. Il s'agit là d'une sorte d'exploitation des informations contenues dans la population courante de génomes. En revanche, cet opérateur, combiné avec le processus de sélection qui élimine des individus, tend à appauvrir le stock de primitives disponibles. Il est donc nécessaire de disposer d'un autre opérateur introduisant des nouveautés dans ce stock de primitives. C'est le rôle de l'opérateur de *mutation* qui, par la modification aléatoire de bits, conserve un taux de renouvellement, donc de diversité, des génomes. La mutation joue ainsi un rôle d'exploration.

La proportion de ces deux opérateurs dans l'étape de modification d'une génération $P_g(t)$ pour passer à la suivante est sujette à controverse. Pour John Holland, l'opération de croisement est fondamentale puisqu'elle permettrait l'émergence progressive de primitives de plus en plus intéressantes dans le contexte de l'environnement courant. L'opération de mutation ne serait là que pour conserver une source de diversité nécessaire. En revanche, les promoteurs des stratégies d'évolution ont le discours inverse, minimisant le rôle du croisement pour se concentrer sur celui de la mutation, ce qui revient à implémenter une sorte d'algorithme de gradient stochastique. Qui a raison ?

Il est utile de se reporter au *no-free-lunch theorem* qui sera débattu dans le chapitre 17. Ce théorème affirmant qu'aucune méthode inductive n'est meilleure qu'une autre indépendamment d'une classe de problèmes donnée a une contrepartie pour ce qui concerne les méthodes d'optimisation par recherche dans un espace de solutions (voir [Wol97]). Ici aussi, aucune méthode d'optimisation n'est meilleure en moyenne sur tous les problèmes d'optimisation possibles. Il en résulte que chaque méthode est adaptée à certaines classes de problèmes et inopérante pour d'autres classes de problèmes. La controverse entre partisans d'une forte proportion d'opérations de croisement et ceux favorisant l'usage de la mutation est donc stérile. Ce qui est intéressant est d'essayer d'identifier les types de problèmes pour lesquels chaque méthode est la plus performante. Dans l'état actuel des connaissances, il semble que l'usage important de l'opération de croisement soit intéressant lorsque la fonction à optimiser est relativement continue, ne présentant pas trop de discontinuités.

À ceux que cette caractérisation très qualitative ne satisfait pas, nous conseillons de consulter

les travaux récents portant sur la notion de « rugosité » du paysage de la fonction à optimiser (voir par exemple [Ang98]).

Pour terminer sur ce problème, il est à noter que les techniques les plus récentes utilisent une adaptation dynamique de la proportion de chaque opérateur en fonction des informations glanées lors de l'exploration de l'espace de recherche. Cela facilite aussi l'adaptation dans des environnements changeant avec le temps.

8.3.5 La sélection

Les opérateurs étudiés dans la section précédente fonctionnent dans l'espace \mathcal{G} des génotypes. La sélection, qui consiste à calculer la nouvelle population d'hypothèses, s'effectue dans l'espace \mathcal{H} sous la pression de l'environnement décrit dans l'espace des entrées \mathcal{X} . On peut y distinguer trois phases : une phase d'*évaluation* dans laquelle chaque hypothèse $h \in \mathcal{H}$ est évaluée (par exemple en mesurant son risque empirique), une phase de *sélection* proprement dite où l'on ne retient qu'une partie de la population courante d'hypothèses, et une phase de *remplacement* qui remplace la population courante $P_{\mathcal{H}}(t)$ par une nouvelle population $P_{\mathcal{H}}(t+1)$.

Il est tentant d'évacuer rapidement la *phase d'évaluation* des hypothèses. Après tout il n'y a rien là de nouveau : il s'agit seulement d'évaluer la qualité ou la performance de chaque hypothèse dans le contexte des données d'apprentissage. Dans le contexte des algorithmes opérant sur le génotype, on parle de *fonction d'évaluation* ou de *fonction de performance* quand on n'adopte pas directement le terme anglais de *fitness function*. Nous voulons cependant souligner l'importance de cette étape dans le contexte des algorithmes opérant sur le génotype. En effet, ces algorithmes impliquent généralement l'évolution de populations d'hypothèses, ce qui signifie que le plus souvent de très nombreuses hypothèses doivent être testées. Si l'évaluation d'une hypothèse est coûteuse, comme c'est fréquemment le cas (par exemple dans la recherche d'une structure mécanique devant remplir un certain cahier des charges, ou dans la recherche d'un programme de commande de robot), alors l'évaluation d'une population d'hypothèses peut être, et de loin, l'étape la plus exigeante en ressources computationnelles. Dans ce cas, on peut utiliser, si c'est possible et avec circonspection, une approximation de la fonction d'évaluation pour accélérer le calcul, surtout pour les premières générations. Une fois que toutes les hypothèses de la population courante ont été évaluées, il faut décider quelles hypothèses méritent attention et peuvent servir pour le calcul de la génération suivante. C'est le rôle de l'étape de sélection.

8.3.5.1 L'étape de sélection

Une fois que la qualité des individus est déterminée grâce à la *fonction de performance* ϕ , il reste à décider quels individus seront sélectionnés pour le calcul de la génération suivante par l'utilisation d'opérateurs génétiques, et quels individus seront conservés ou au contraire éliminés. C'est ce que réalise l'étape de sélection.

D'un certain côté, l'étape de sélection joue un rôle de méthode d'exploitation puisqu'elle utilise l'information rendue disponible par l'évaluation de la population courante des hypothèses pour décider où doit porter l'attention dans la génération suivante.

Conformément à la théorie darwinnienne de l'évolution, la sélection doit tendre à favoriser les meilleurs individus d'une population pour en faire les « parents » de la génération suivante, c'est-à-dire ceux qui transmettront du matériel génétique. Orientant l'exploration de \mathcal{G} , ils contrôleront aussi l'exploration de \mathcal{H} , avec l'espoir de la conduire dans les régions les plus intéressantes. Il faut cependant régler avec soin la pression sélective, c'est-à-dire la tendance à ne conserver que les meilleurs individus. Si en effet celle-ci est trop élevée, le risque est grand d'une perte de diversité dans la population d'hypothèses, ce qui conduit à une convergence prématurée. Inversement,

une pression sélective insuffisante ne conduit pas à une convergence. Plusieurs techniques ont été proposées pour régler ce compromis. Nous en présentons ici trois parmi les plus classiques.

1. La **sélection proportionnelle à la performance** mesurée. Chaque hypothèse $h \in P_{\mathcal{H}}(t)$ est évaluée par la fonction de performance ϕ (phi comme *fitness*). La performance proportionnelle de chaque hypothèse est alors définie comme :

$$\phi_{prop}(h_i) = \frac{\phi(h_i)}{\sum_{h_j \in P_{\mathcal{H}}(t)} \phi(h_j)}$$

Selon la sélection proportionnelle à la performance, la probabilité qu'un individu soit sélectionné pour participer à la production de la génération suivante est égale à sa performance proportionnelle $\phi_{prop}(h)$. Par analogie avec la roulette des casinos, on parle souvent de *sélection par roulette* (voir figure 8.5).

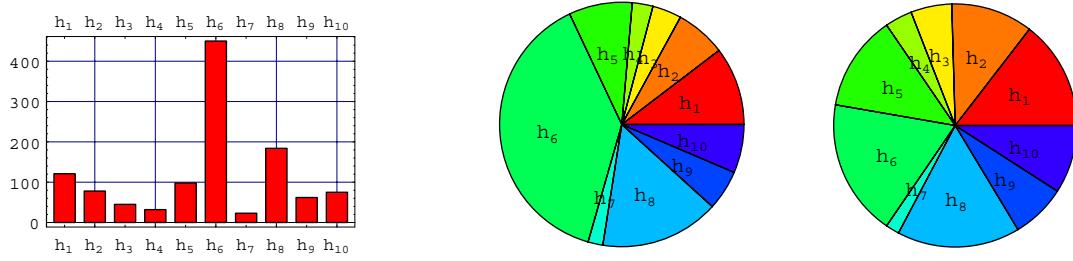


FIG. 8.5 – On considère ici une population de dix hypothèses dont la performance est figurée dans le graphique de gauche. Dans la sélection par roulette (premier camembert), à chaque individu est associé un rayon de la roulette dont l'angle est proportionnel à sa performance absolue. Le tirage au hasard selon la roulette donne une espérance de tirage de chaque individu proportionnelle à sa performance absolue. Dans la sélection par roulette proportionnelle au rang (second camembert), à chaque individu est associé un rayon de la roulette dont l'angle est proportionnel à son rang dans la population, le meilleur individu étant crédité du plus grand nombre, le second de ce nombre moins un, le dernier de un. Le tirage au hasard selon la roulette donne une espérance de tirage de chaque individu proportionnelle à ce nombre.

2. La **sélection proportionnelle au rang**. Un inconvénient de la sélection proportionnelle à la performance est que si un ou plusieurs « superindividu(s) » apparaît(e)s, de performance très supérieure aux autres, ils risquent de complètement monopoliser le pool génétique dont sera issu la prochaine génération. Il arrive alors fréquemment que la population stagne dans un optimum local. Pour éviter ce risque de perte de diversité prématuée, on utilise souvent la sélection proportionnelle au rang. Dans ce cas, les individus sont ordonnés dans l'ordre de leurs performances respectives, et la probabilité de participer à la construction de la prochaine génération est déterminée par le rang et non directement par la performance. Par exemple, on utilise une roulette dans laquelle chaque individu est associé à une part proportionnelle à $\frac{rang(h_i)}{N(N-1)/2}$, où N est la taille de la population. L'effet obtenu est celui d'une pression sélective adoucie lorsque la variance des performances est grande, tandis qu'elle est au contraire accentuée si la variance est faible (voir figure 8.5).

3. La sélection par tournoi. Les méthodes de sélection décrites ci-dessus exigent l'évaluation de toutes les hypothèses de la population. Une autre méthode peut éviter cela, qui de plus se prête bien à une parallélisation des calculs. Supposons que l'on ait besoin de sélectionner μ individus pour engendrer la génération suivante, l'idée est alors d'organiser μ tournois de $t \leq N$ individus chacun. On retient alors le meilleur individu de chaque tournoi. La taille k de ces tournois est un paramètre permettant de contrôler la pression sélective: plus k est grand, plus celle-ci est grande. Dans la pratique, on utilise souvent des valeurs de k comprises entre 2 et 10. Un autre atout de cette méthode est qu'elle est peu sensible aux erreurs de la fonction de performance ϕ .

Une fois que les individus les plus aptes ont été sélectionnés, ils sont utilisés comme entrées des opérateurs génétiques afin de calculer des individus neufs. Supposons que μ individus aient ainsi été retenus: un cas typique est d'utiliser environ $\mu/2$ individus pour des croisements, $\mu/20$ pour de la mutation et le reste pour une simple recopie. Bien sûr ces proportions peuvent varier, et même considérablement. Ainsi dans les *stratégies d'évolution*, le croisement n'est pas employé, tandis que les μ individus sélectionnés sont utilisés pour l'opération de mutation.

8.3.5.2 L'étape de remplacement

Après que la sélection a retenu μ parents et que ceux-ci ont produit λ descendants, il faut décider de quels individus sera composée la population suivante, sachant qu'en général la taille N de la population reste fixe. Plusieurs variantes sont possibles suivant que des individus de la population précédente sont admis à perdurer dans la nouvelle population ou non.

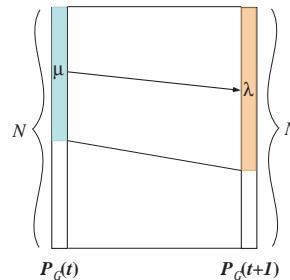


FIG. 8.6 – Le schéma général du remplacement d'une population par la suivante. μ parents sont sélectionnés dans la population courante pour engendrer λ individus qui figureront dans la population suivante. Le reste des individus nécessaires à la complémentation de la population est obtenu par des méthodes variées.

Une méthode extrême consiste à remplacer entièrement la génération précédente par les descendants calculés. On parle alors de *remplacement* (μ, λ) où les μ individus de la génération suivante (ici $\mu = N$) sont les meilleurs des λ individus obtenus par les opérateurs génétiques. L'inconvénient de cette méthode est qu'elle risque de perdre des individus excellents d'une génération et de les remplacer par des individus plus médiocres. C'est pourquoi une autre méthode est plus souvent employée, le *remplacement* ($\mu + \lambda$), qui consiste à tirer les N nouveaux individus parmi les μ parents sélectionnés plus les λ descendants calculés.

Une alternative à ces méthodes impliquant le remplacement d'une génération par la suivante est la technique de *génération par remplacement local (steady-state)*. Le principe consiste à sélectionner une sous-population de la population totale puis de déterminer par roulette ou par tournoi un ou des parents dans cette sous-population pour engendrer les remplaçants de cette sous-population, en respectant une population totale constante. Chaque génération n'implique

donc que le remplacement de certains individus et non le remplacement de toute la population. L'intérêt de cette technique est de se prêter aisément à une implémentation asynchrone et décentralisée.

8.3.6 Le théorème des schémas : une explication de la puissance des AG ?

Les algorithmes opérant sur le génotype découpent le problème de l'expression des hypothèses de celui de son exploration en déléguant celui-ci à un autre espace doté d'opérateurs propres : l'espace des génotypes. Ces algorithmes ont également une source d'inspiration dans les modèles de l'évolution. Depuis leur conception, les algorithmes opérant sur le génotype ont été testés sur de très nombreux domaines d'application et revendiquent des succès parfois spectaculaires sur des problèmes réputés difficiles. Il est temps de s'interroger sur la source de la puissance de ces algorithmes, ce qui devrait permettre aussi, dans l'esprit du *no-free-lunch theorem* (voir 17.4.1), de mieux cerner les familles de problèmes pour lesquels ils sont les plus adaptés.

Depuis quelques années, plusieurs travaux de recherche importants ont été dédiés à l'étude théorique du fonctionnement des algorithmes opérant sur le génotype (voir par exemple [Cer95, Kal99, Rud97, Vos99] ...). Ces travaux utilisent principalement des modélisations par chaînes de Markov, par martingales, ou des concepts tels que la rugosité de l'espace de recherche. Nul doute que ces travaux défrichent un terrain qui se révélera fécond et qu'un spécialiste des algorithmes opérant sur le génotype doit connaître. Cependant, il nous semble intéressant de consacrer l'espace limité de ce chapitre à une analyse théorique déjà ancienne : le *théorème des schémas* dû à Holland en 1975. En effet, même si elle a été depuis l'objet de nombreuses études critiques et semble dépassée sur certains points, elle est encore la source de profondes interrogations.

Le point de départ de cette étude théorique réside dans un résultat de la théorie de Markov, à savoir que le nombre de tests nécessaires à l'obtention d'un optimum global par une recherche aléatoire croît exponentiellement avec la dimension L de l'espace de recherche quelle que soit la fonction de performance cible. Qu'est-ce qui fait alors la différence avec des algorithmes utilisant le génotype ? Qu'est-ce qui peut expliquer leur puissance apparente ?

L'une des motivations de John Holland était de montrer pourquoi l'opérateur de croisement est essentiel dans le fonctionnement des algorithmes génétiques. L'idée fondamentale est que les génotypes associés à des hypothèses contiennent dans leur code des *primitives de construction* (*building blocks*). Dans le cadre des algorithmes génétiques, une primitive peut être n'importe quelle sous-séquence de chaîne de bits qui se présente dans la population. Mais cette notion s'étend aussi aux algorithmes utilisant d'autres types de représentations : des sous-arbres dans le cas de la programmation génétique ou des portions d'automates dans le cas de la programmation évolutive. L'hypothèse est que la possession dans leur code génétique de bonnes primitives augmente la performance des individus qui en sont dotés (comme la possession d'un cœur augmente notamment le domaine d'action des organismes qui en sont dotés). De ce fait, ces individus tendent à être davantage sélectionnés pour la reproduction de la nouvelle génération et par conséquent ces primitives tendent à se multiplier et à se répandre dans la population (le théorème des schémas montre que cet accroissement des bonnes primitives est exponentiel en cours d'évolution). Le croisement permettant la combinaison de ces primitives en super-primitives conduirait alors à l'émergence de superindividus.

Plus formellement, on recherche une hypothèse optimale *via* l'exploration de l'espace des génotypes \mathcal{G} , ce qui revient à chercher un génotype optimal. L'hypothèse fondamentale de Holland est qu'un génotype optimal est une *superposition* de parties de génotypes qu'il appelle des schémas, qui correspondent chacune à des propriétés particulières des hypothèses associées. Ainsi, un schéma est une primitive de construction dans la mesure où le trait qu'il code dans \mathcal{H} corres-

pond à une propriété mesurable et que c'est l'*addition* de telles propriétés qui rend une hypothèse performante. Il est crucial de noter que cette hypothèse fondamentale est nullement triviale et correspond en fait à une contrainte très forte sur la relation entre \mathcal{H} et \mathcal{G} . Cela signifie en effet que le codage des hypothèses dans \mathcal{G} respecte une indépendance entre les propriétés pertinentes (pour la performance) des hypothèses, la superposition de deux schémas étant associée à l'addition de deux propriétés mesurables dans \mathcal{H} . Trouver un tel code n'est généralement pas une tâche aisée. Nous allons voir que le théorème des schémas, qui découle de cette hypothèse fondamentale, ajoute encore des contraintes sur ces schémas si l'on veut que les algorithmes opérant sur le génotype fonctionnent. La discussion qui suit s'applique aux algorithmes génétiques qui opèrent sur des chaînes de bits de longueur fixe. L'extension à d'autres types de représentations a fait l'objet de travaux mais se heurte à des difficultés que nous soulignerons dans les sections correspondantes.

Soit L la longueur des chaîne de bits de \mathcal{G} .

Définition 8.1 (Schéma)

Un schéma s est une chaîne sur $\{0, 1, *\}^L$ où la signification du symbole * est 0 ou 1. Il y a par conséquent 3^L schémas définis sur \mathcal{G}

Un schéma correspond ainsi à un sous-espace projeté de \mathcal{G} .

Exemple 8.1

Par exemple, le schéma $s = 0*10$ défini sur un espace de chaînes de bits de longueur 4 est appariable à l'ensemble des chaînes de bits contenant exactement 0010 et 0110.

Inversement, une chaîne de bits g est appariable à plusieurs schémas : tous ceux qui correspondent aux sous-espaces de \mathcal{G} contenant g . Par exemple, la chaîne de bits 0100 est contenue dans 2^4 schémas (c'est-à-dire dans les sous-espaces de \mathcal{G} associés), dont par exemple **** (dimension 4), *1** (dimension 3), 0*0* (dimension 2) ou encore 01*0 (dimension 1). On peut donc considérer qu'un individu g de \mathcal{G} est représentatif d'un certain nombre de schémas, dont les traits correspondants sont exprimés dans l'hypothèse h associée à g . Appelons $n(s, t)$ le nombre de schémas s représentés par une population de génotypes à l'instant t . Le théorème des schémas décrit l'espérance de $n(s, t + 1)$ en fonction de $n(s, t)$ et d'autres propriétés de la population et des paramètres de l'algorithme.

L'évolution de la population dans un algorithme génétique dépend de l'étape de sélection, puis des opérations de croisement et de mutation. Considérons d'abord l'effet de la sélection. Par abus de langage, on parlera de la performance $\phi(g)$ d'un génotype g , quand en toute rigueur il s'agit de la performance mesurable de l'hypothèse associée h . Soit donc $\bar{\phi}(t)$ la performance moyenne des individus de la population $P_{\mathcal{G}}(t)$ à l'instant t :

$$\bar{\phi}(t) = \frac{1}{N} \sum_{g \in P_{\mathcal{G}}(t)} \phi(g)$$

D'après la règle de sélection proportionnelle à la performance, la probabilité pour un génotype g_i d'être sélectionné est :

$$Pr(g_i) = \frac{\phi(g_i)}{\sum_{g_j \in P_{\mathcal{G}}(t)} \phi(g_j)} = \frac{\phi(g_i)}{N \bar{\phi}(t)} \quad (8.1)$$

La probabilité de sélectionner un génome représentatif d'un schéma s est alors :

$$Pr(g \in s) = \sum_{g \in s \cap P_G(t)} \frac{\phi(g)}{N\bar{\phi}(t)} = \frac{\phi(s, t)}{N\bar{\phi}(t)} n(s, t) \quad (8.2)$$

où $\phi(s, t)$ est la performance moyenne des génomes de la population $P_G(t)$ représentatifs du schéma s :

$$\phi(s, t) = \frac{\sum_{g \in s \cap P_G(t)} \phi(g)}{n(s, t)}$$

L'équation 8.2 vaut pour un génome. L'espérance d'appartenance au schéma s pour l'ensemble des individus de la population $P_G(t)$ résulte de N fois le même tirage indépendant, d'où la valeur de l'espérance :

$$E[n(s, t + 1)] = \frac{\phi(s, t)}{\bar{\phi}(t)} n(s, t) \quad (8.3)$$

Si l'on regarde seulement l'effet de la sélection d'une génération à la suivante, l'équation 8.3 montre que l'espérance du nombre de génomes représentatifs d'un schéma s à l'instant $t + 1$ est proportionnelle à la performance moyenne de ce schéma à l'instant t et inversement proportionnelle à la performance moyenne de la population de génomes à l'instant t . Les schémas dont la valeur moyenne est supérieure à la performance moyenne générale devraient donc voir le nombre de leurs représentants augmenter au cours des générations. Inversement, les schémas associés à des performances médiocres devraient tendre à disparaître. Globalement, l'effet des algorithmes génétiques devrait être de révéler et de favoriser les primitives de construction intéressantes pour la tâche courante.

Cependant, cet effet sélectif n'est pas suffisant pour construire des superindividus. Il faut tenir compte des opérateurs de croisement et de mutation. Or ceux-ci, à côté de leurs éventuels effets bénéfiques, ont aussi des effets destructeurs. De fait, sous l'effet d'une mutation, un génome peut ne plus appartenir à un schéma donné. La probabilité que cela arrive dépend du nombre de bits définissant un schéma. Une opération de croisement (ici on ne considère que les croisements à un point) peut de la même manière produire des descendants n'appartenant plus à un schéma donné. Il suffit pour cela que le point de croisement se trouve situé « à l'intérieur » du schéma considéré.

Pour énoncer complètement le théorème des schémas, il nous faut donc deux définitions exprimant des caractéristiques des schémas importantes lorsque les opérateurs génétiques sont employés.

Définition 8.2 (Ordre d'un schéma)

On appelle ordre $o(s)$ d'un schéma s le complément à L de la dimension du sous-espace correspondant. C'est aussi le nombre de caractères dont la valeur est spécifiée dans s .

Exemple 8.2

C'est ainsi que le schéma `**0**` est d'ordre 1, alors qu'il représente un sous-espace de dimension 4 de $\{0, 1, *\}^5$ et le schéma `10 * 11` d'ordre 4 .

Définition 8.3 (Longueur caractéristique d'un schéma)

On appelle longueur caractéristique $d(s)$ d'un schéma s , dans sa représentation comme une chaîne de L caractères de l'alphabet $\{0, 1, *\}$, le nombre de bits entre le symbole non égal à `*` le plus à gauche dans la chaîne et celui le plus à droite, augmenté de 1.

Exemple 8.3

C'est ainsi que $d(1 * * * 1) = 4$, $d(*0 * 1*) = 2$, $d(10 * 1*) = 3$ et $d(* * 0 * *) = 1$.

En notant p_{crois} la probabilité d'utilisation de l'opérateur de croisement et p_{mut} la probabilité de mutation d'un bit, on obtient alors l'équation suivante, appelée aussi *théorème des schémas*.

Théorème 8.1 (Le théorème des schémas (Holland, 1975))

L'espérance du nombre de génomes représentants un schéma s suit l'équation :

$$E[n(s, t + 1)] = \frac{\phi(s, t)}{\phi(t)} n(s, t) \left(1 - p_{crois} \frac{d(s)}{L - 1} \right) (1 - p_{mut})^{o(s)} \quad (8.4)$$

Le premier terme de droite décrit l'effet de la sélection, conformément à l'équation 8.3. Le deuxième décrit l'effet destructeur du croisement. Il exprime la probabilité de survivance à l'effet d'un opérateur de croisement. Plus la longueur caractéristique d'un schéma est grande, plus la probabilité qu'un opérateur de croisement détruisse l'appartenance à ce schéma d'un descendant est grande. Le troisième terme correspond à l'effet destructeur des mutations, dont la probabilité dépend de l'ordre du schéma considéré. Ainsi les effets destructeurs des opérateurs de croisement et de mutation s'accroissent avec l'ordre et la longueur caractéristique des schémas. Holland en conclut que l'effet des algorithmes génétiques est de favoriser l'apparition des représentants des schémas les plus performants, et ceci d'autant plus que ceux-ci sont courts (ordre et longueur caractéristique faibles).

Plusieurs remarques peuvent être faites :

1. D'après ce théorème, l'idéal serait de disposer d'un codage des génomes tel que les primitives utiles soient décrites par des chaînes de bits courtes. Malheureusement, cela revient à connaître à l'avance le codage adéquat, c'est-à-dire les propriétés du problème que l'on cherche à résoudre. Les détracteurs de ce théorème se sont ingénier à concevoir des problèmes dits « trompeurs » (*deceptive problems*) qui ne satisfont pas les codages usuels et mettent donc les algorithmes génétiques en défaut. On pourra noter ici que certains auteurs, inspirés aussi par la biologie, ont proposé des *opérateurs d'inversion* permettant le réarrangement dans l'espace des schémas afin de pouvoir réduire leur longueur caractéristique et de les soustraire aux effets destructeurs du croisement.
2. Le théorème des schémas est insuffisant car, bien qu'il prenne en compte l'effet destructeur des opérateurs génétiques, il ne traduit pas leurs éventuels effets bénéfiques, en particulier ceux concernant le croisement qui permet la combinaison de plusieurs schémas en superschémas. Cette étude-là en est encore à ces prolégomènes.
3. Une autre source d'insuffisance du théorème des schémas réside dans son incapacité à prendre en compte ce que l'on appelle l'*épistasie*, qui signifie que certaines parties du génome peuvent avoir une influence sur l'expression d'autres parties du génome. Plus généralement, toutes les non-linéarités dans le fonctionnement ou l'expression du génome échappent à l'analyse par schémas.
4. Une idée profonde de Holland est que les algorithmes génétiques, en calculant des populations successives de génomes, effectuent de fait une *recherche implicite dans l'espace des schémas*, pour découvrir les plus intéressants. Si c'est là effectivement la source de puissance des algorithmes génétiques, alors il faut favoriser cette recherche implicite qui se réalise grâce à l'appartenance des génomes à plusieurs schémas, lesquels se trouvent ainsi mis en compétition. Il faut donc maximiser le nombre de schémas qui sont représentés par une population de génomes de taille donnée. Cette remarque conduit à préférer les

alphabets binaires, d'où le choix de la représentation prônée par Holland. En effet, par exemple, la chaîne 0101 est représentative de 2^4 schémas dans un alphabet binaire, alors que le nombre total de schémas est de 3^L . Dans un alphabet n -aire, la chaîne 0101 est toujours représentative de 2^4 schémas, tandis que leur nombre total est de n^4 . On voit que l'exploration implicite permise par une population de taille donnée est ainsi plus efficace dans le cas d'un alphabet binaire.

L'étude décrite ci-dessus concerne des représentations séquentielles de taille fixe. Que se passe-t-il dans le cas de l'utilisation de représentations dont la taille peut varier et dont les primitives peuvent être réarrangées dans le génome? On a encore beaucoup de mal à l'analyser, mais il existe plusieurs propositions très intéressantes d'algorithmes utilisant des représentations alternatives et qui ont été largement testées sur de nombreux problèmes.

8.4 Les stratégies d'évolution

La technique dite des « stratégies d'évolution » a été initiée par Fogel, Owens et Walsh dans les années 1960 (voir [FOW66]). Elle consiste à utiliser une représentation par automates à états finis³ pour l'espace \mathcal{G} des génotypes et l'opérateur de mutation pour modifier les expressions de cet espace. Dans leur version originale, les stratégies d'évolution furent utilisées pour des tâches de prédiction de séquences.

La méthode démarre en produisant une population initiale d'automates à états finis. Ceux-ci sont parcourus en fonction de l'état courant et de l'entrée correspondant au caractère courant de la séquence. L'évaluation de l'automate tient compte de l'accord ou non de la prédiction faite alors par l'automate par rapport au caractère suivant de la séquence. De cette manière, chaque automate est évalué en fonction de la séquence entière à prédire. La sélection utilise une roulette proportionnelle à la performance. Les descendants de la génération courante sont calculés à partir des parents sélectionnés en leur appliquant des mutations qui peuvent ajouter des états, en retirer, changer les transitions entre états, changer l'état initial et changer les caractères de sortie. Très souvent, la génération suivante est obtenue par un procédé de remplacement élitiste consistant à conserver la meilleure moitié de la génération courante et en complétant par les descendants obtenus à partir de cette meilleure moitié. Souvent aussi, le taux de mutation est réduit au fur et à mesure des générations.

Les développements récents de cette technique ont élargi les recettes employées, mais en conservant la mutation comme seul opérateur génétique (voir [Fog98, Fog99]).

Les figures 8.7 et 8.8 illustrent l'évolution d'une population de cinquante automates à états finis ayant au maximum vingt états sur une tâche de prédiction de la séquence :

1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1

La figure 8.7 montre la population initiale, tandis que la figure 8.8 montre l'évolution du meilleur automate toutes les dix générations depuis la génération 10 jusqu'à la génération 200. (Voir [Jac01], ch.6 pour les détails).

8.5 La programmation génétique

Est-il possible d'envisager de produire automatiquement des programmes par un mécanisme d'évolution simulée? Ici, l'espace \mathcal{H} des hypothèses deviendrait un espace de programmes, la performance de chacun d'eux étant mesurable par test dans un environnement représentatif des

3. Voir le chapitre 7.

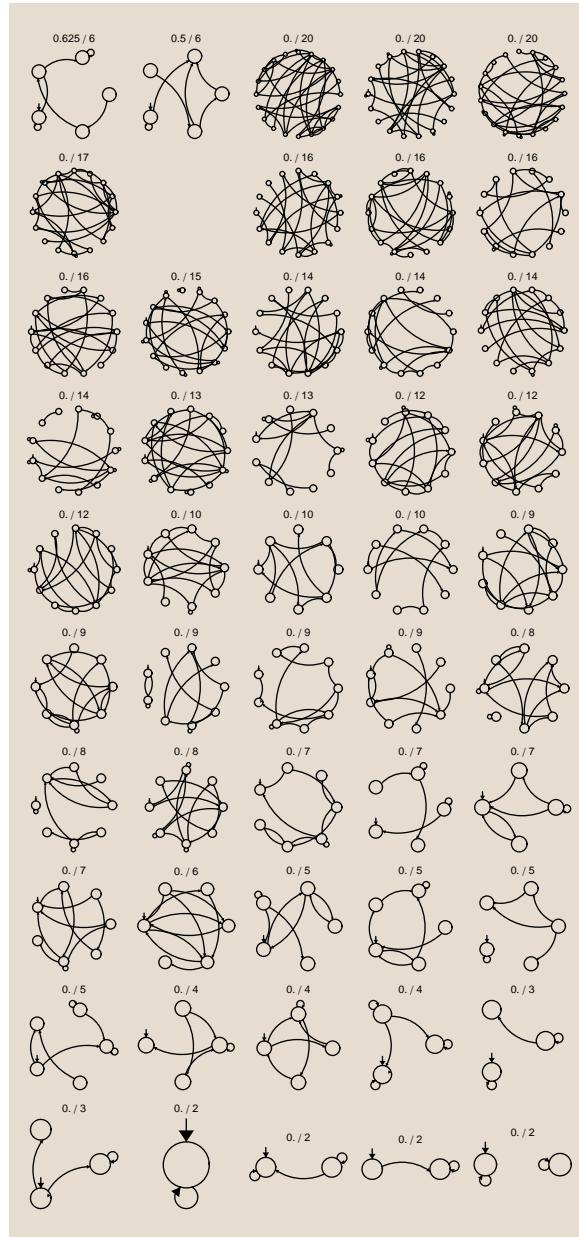


FIG. 8.7 – La génération 0 d'automates à états finis.

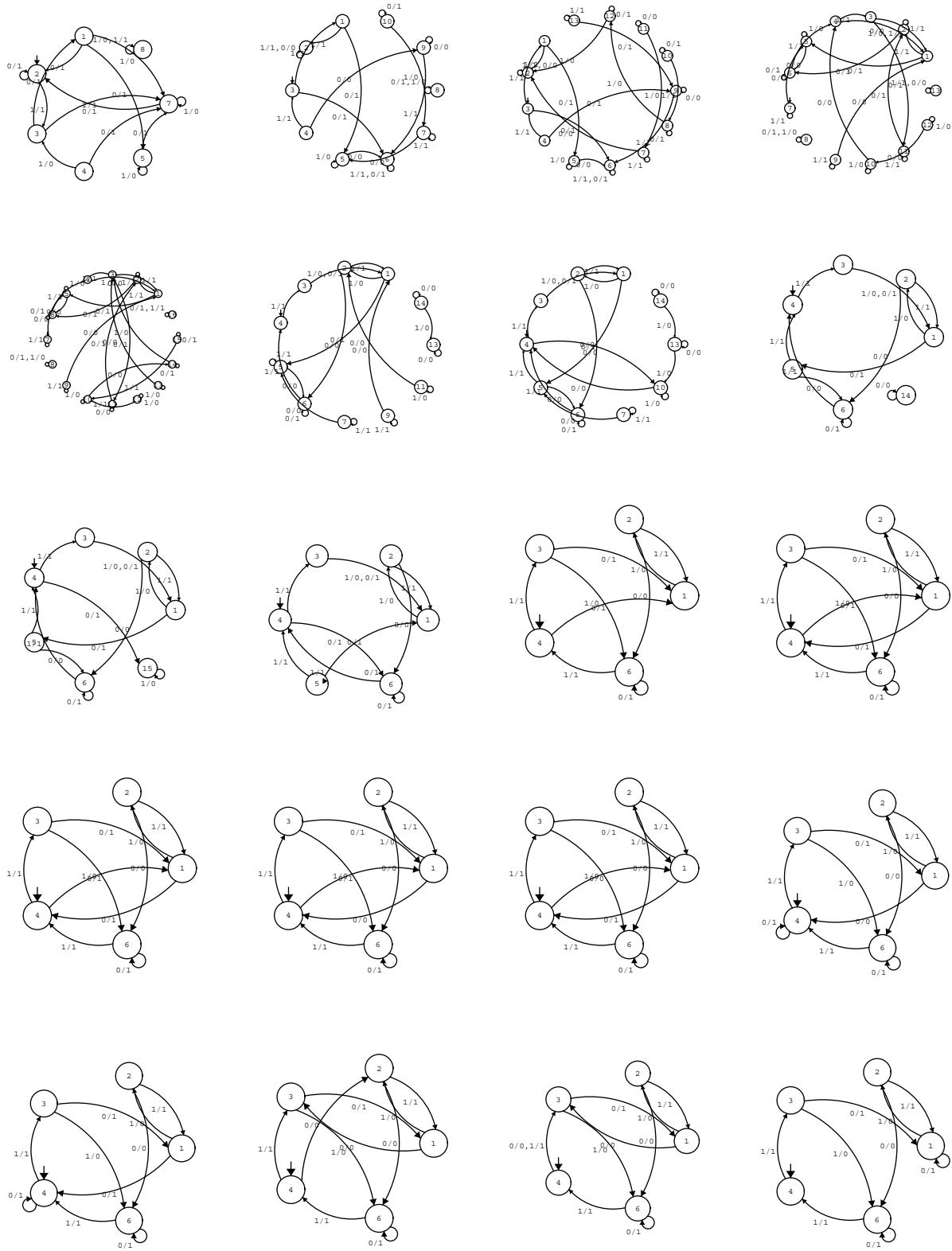


FIG. 8.8 – Le meilleur automate toutes les dix générations de la génération 10 à la génération 200. Dès la génération 70, le meilleur automate prédit parfaitement la séquence. Le reste de l'évolution conduit à une simplification du meilleur automate.

tâches à réaliser. L'idée est séduisante, elle semble cependant complètement irréalisable. Tout programmeur a déjà eu l'occasion de pester contre une virgule ou un point-virgule mal placé responsable du plantage d'un programme, alors comment une exploration au hasard, ou presque, de l'espace des expressions possibles pourrait parvenir à produire un programme valide, encore plus d'un programme solution, et qui plus est efficace?!

Encore une fois la solution réside dans une représentation adéquate des programmes se prêtant bien à l'utilisation d'opérateurs d'exploration efficaces. En intelligence artificielle, dire ceci revient évidemment à énoncer une tautologie. Personne n'a de recette pour trouver une telle représentation ni les opérateurs associés. Cependant, toute une école de chercheurs, à la suite de pionniers tels que Cramer [Cra85] et Koza [Koz92, Koz94, KA99], a proposé une nouvelle approche qui a été testée de manière prometteuse sur une grande variété de problèmes. L'idée fondamentale est de s'appuyer sur une représentation des programmes par arbres et d'utiliser des langages de programmation dont la syntaxe est simple et telle que des manipulations syntaxiques également simples produisent encore des programmes licites. Cette idée, déjà largement exploitée par Douglas Lenat dans son programme AM (Automated Mathematician [Len78]), conduit à s'intéresser à des langages de programmation tels que Lisp ou des dérivés comme Scheme ou Mathematica.

8.5.1 La représentation des programmes

La différence essentielle entre les algorithmes génétiques et la programmation génétique est que dans ces derniers les structures de programmation ne sont pas codées par des génomes linéaires, mais par des termes ou des expressions symboliques correspondant à un langage de programmation. Les unités objets de mutation ou de recombinaison ne sont plus des caractères ou des sous-séquences de chaînes, mais des modules fonctionnels (à la Lisp) représentés par des arbres. Une telle représentation conduit à une redéfinition des opérateurs génétiques. Ainsi, les primitives de représentation sont maintenant des termes (feuilles de l'arbre) ou des sous-termes (sous-arbres) qui peuvent être soit remplacés (mutation) soit échangés entre des arbres représentant des programmes (recombinaison). En comparaison avec les algorithmes utilisant des génomes linéaires, cela ouvre des possibilités inédites. Par exemple, la taille des génomes n'est plus fixée ni même en principe limitée en taille ou en profondeur. Les adaptations possibles des structures de programmes sont ainsi potentiellement illimitées, correspondant à de grandes variations tant dans la sémantique que dans les aspects algorithmiques des programmes. Certains pensent même que cela ouvre des perspectives radicalement nouvelles par rapport à la biologie dont le matériel génétique est codé sur des chromosomes linéaires.

8.5.1.1 L'ensemble des terminaux et des fonctions

En programmation génétique, les fonctions et les terminaux sont les primitives servant à décrire les programmes. Qualitativement, on peut dire que les *terminaux* fournissent des valeurs au programme tandis que les *fonctions* traitent ces valeurs. Les terminaux correspondent aux feuilles des arbres, tandis que les fonctions correspondent aux noeuds internes.

Définition 8.4 (Ensemble des terminaux)

L'ensemble des terminaux est constitué par les entrées des programmes, les constantes définies a priori et les fonctions d'arité nulle ayant des effets de bord calculés en cours d'exécution.

Les terminaux retournent une valeur numérique sans recevoir d'arguments en entrée. En général ces terminaux sont constitués des valeurs numériques rencontrées dans l'échantillon

d'apprentissage en plus de constantes jugées utiles par le concepteur. D'autres valeurs numériques pourront être calculées par l'usage de fonctions prenant des terminaux en argument.

Définition 8.5 (Ensemble des fonctions)

L'ensemble des fonctions est constitué des déclarations, opérateurs et fonctions disponibles pour le système.

Exemple 8.4

Voici quelques exemples de fonctions qui peuvent être utilisées en programmation génétique.

- Fonctions booléennes. *Exemples : AND, OR, NOT, XOR*
- Fonctions arithmétiques. *Exemples : PLUS, MINUS, MULTIPLY, DIVIDE*
- Fonctions transcendantes. *Exemples : Fonctions trigonométriques et logarithmiques.*
- Formes conditionnelles. *Exemples : IF ... THEN ... ELSE*
- Structures itératives. *Exemples : boucles REPEAT, WHILE, DO, ...*
- Sous-routines. *Le domaine des fonctions utilisables est en fait bien plus considérable que les fonctions prédéfinies d'un langage et peuvent inclure n'importe quelle primitive définie par le programmeur, par exemple en réponse à un problème spécifique. Nous en verrons un exemple dans la section 8.5.5 à propos d'un robot simulé.*

8.5.1.2 Le choix de l'ensemble des terminaux et des fonctions

Le choix de l'ensemble des terminaux et des fonctions doit obéir à quelques critères. Il doit être assez riche pour permettre l'expression d'une solution au problème. Mais il ne doit pas l'être trop sous peine de rendre trop coûteuse l'exploration de l'espace de recherche. En général, il est préférable de commencer avec un ensemble restreint et de l'enrichir si l'expérience en fait sentir le besoin. Une autre propriété est nécessaire : la propriété de *clôture*. Chaque fonction de l'ensemble des fonctions doit en effet pouvoir traiter toutes les valeurs qu'il peut recevoir en entrée. Ceci est très important car les entrées d'une fonction donnée peuvent varier énormément au cours de l'évolution simulée de programmes. Il ne faut pas que cela provoque un arrêt du programme (comme cela peut arriver si une division par zéro est demandée). La propriété de clôture est plus facile à vérifier avec des langages dont la syntaxe est uniforme comme c'est le cas de Lisp, elle exige cependant une certaine attention.

8.5.1.3 Phénotypes et génotypes en programmation génétique

Les algorithmes utilisant le génotype pour la recherche d'une bonne solution ou hypothèse découpent les problèmes d'expressivité du langage de représentation (ceux de coût mémoire ou d'exécution qui sont propres à l'espace \mathcal{H} des phénotypes) des problèmes de recherche dans l'espace des programmes (qui sont du ressort de l'espace \mathcal{G} des génotypes). Dans le cas de la programmation génétique cependant, la distinction entre les deux espaces n'existe généralement pas, et la recherche par évolution simulée s'opère directement dans l'espace \mathcal{H} des programmes exécutables et testables. Les avocats de la programmation génétique estiment par conséquent que la représentation et les opérateurs dont ils disposent sont efficaces directement dans \mathcal{H} .

8.5.1.4 Initialisation d'une population de programmes

Au début d'un cycle d'évolution simulée, il faut produire une population initiale de programmes. Cela est généralement réalisé par un tirage aléatoire d'arbres représentant des programmes. Il est souvent spécifié une profondeur limite ou un nombre maximal de noeuds que

les arbres produits aléatoirement ne peuvent dépasser. Le processus de création des arbres est récursif, avec à chaque pas la création d'un nœud de l'arbre par sélection aléatoire d'une fonction ou d'un terminal. Dans ce dernier cas, une feuille de l'arbre est atteinte. Plusieurs méthodes de génération d'arbres ont été étudiées. Nous renvoyons le lecteur par exemple à [BNKF98] pp.118-122.

8.5.2 Les opérateurs génétiques sur les programmes

Comme pour les algorithmes génétiques, les trois opérateurs les plus employés en programmation génétique sont : la mutation, le croisement et la reproduction.

1. **La mutation.** C'est un opérateur défini de $\mathcal{G} \rightarrow \mathcal{G}$ opérant donc sur un seul génoe et produisant un autre génoe par altération aléatoire du premier. Plus précisément, la mutation sur une représentation par arbre opère en sélectionnant au hasard un sous-arbre et en le remplaçant par un autre sous-arbre engendré aléatoirement, en respectant les limites de profondeur ou de taille définies pour l'initialisation de la population. (Voir figure 8.9).

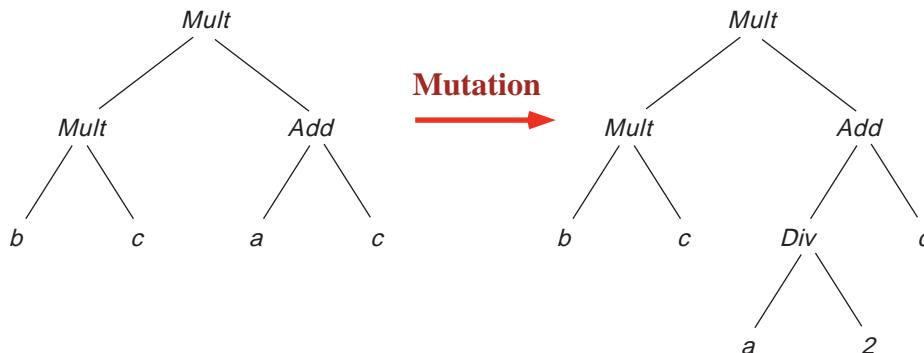


FIG. 8.9 – Exemple d'une opération de mutation.

2. **Le croisement** est un opérateur défini de $\mathcal{G} \times \mathcal{G} \rightarrow \mathcal{G} \times \mathcal{G}$ produisant deux génoes à partir de la donnée de deux génoes en entrée. Il sélectionne au hasard un sous-arbre dans chacun des génoes fournis en entrée et les échange pour produire deux nouveaux génoes (voir figure 8.10). Il est évident que cette opération doit respecter la propriété de clôture définie plus haut (8.5.1.2).
3. **La reproduction** consiste simplement en une recopie d'un individu d'une génération à la suivante.

8.5.3 Évaluation et sélection

L'évaluation des génoes et leur sélection suit les mêmes principes que ceux des algorithmes génétiques. La performance d'un génoe est définie comme la performance mesurée de la solution correspondante dans \mathcal{H} . La sélection des individus utilisés pour produire la génération suivante se fait également selon les mêmes méthodes.

8.5.4 Le fonctionnement

Le fonctionnement de la programmation génétique suit les mêmes étapes que les algorithmes génétiques.

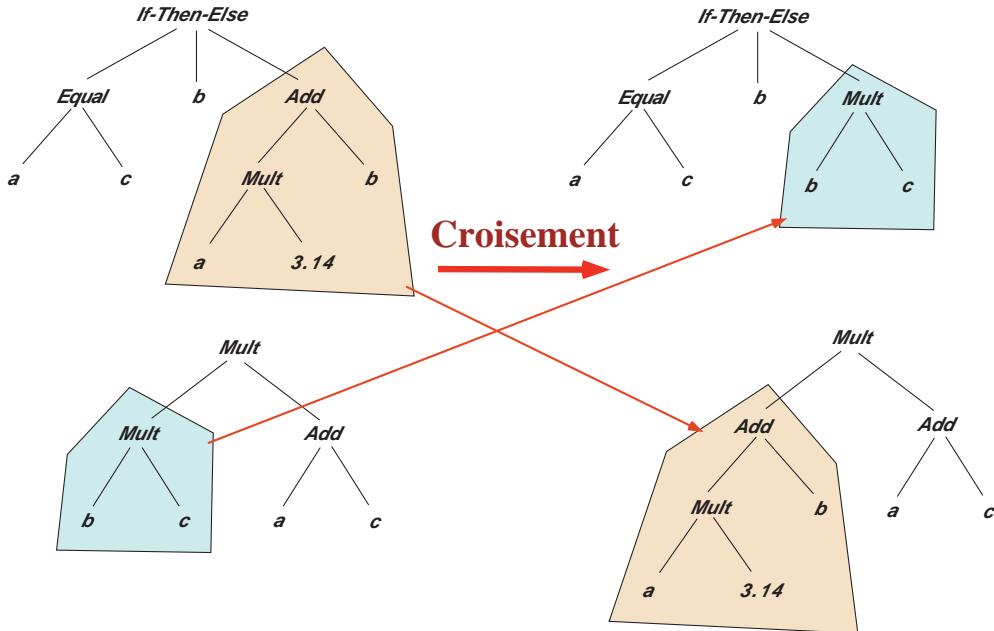


FIG. 8.10 – Exemple d'une opération de croisement.

8.5.5 Illustrations

Nous illustrons brièvement le fonctionnement de la programmation génétique sur deux problèmes. Le premier est un problème d'induction classique. Il s'agit, étant donné un échantillon d'exemples et de contre-exemples décrits dans un espace numérique $\mathcal{X} = \mathbb{R}^d$, de trouver l'équation d'une séparatrice entre les deux classes d'objets. Le second problème concerne l'induction du programme de contrôle d'un robot simulé (censé modéliser une fourmi) dans un environnement simplifié dans lequel se trouve de la « nourriture ».

8.5.5.1 Induction d'une séparatrice entre deux classes

L'expérience rapportée dans cette section concerne l'induction d'une surface séparatrice dans un espace d'exemples appartenant à deux classes et décrits par des descripteurs numériques. Le principe de l'expérience est le suivant: une fonction de séparation cible est choisie aléatoirement par combinaison d'un certain nombre fixé de descripteurs en utilisant un répertoire de fonctions fourni *a priori*. Par exemple, en fixant le nombre de descripteurs à deux et le jeu de fonctions à: $+$, $-$, \times , \div , il est possible de produire la séparatrice: $f(\mathbf{x}) = \frac{x_1 \times x_2}{x_1 + x_2}$. Un ensemble de points de \mathcal{X} tirés aléatoirement sont alors étiquetés par comparaison avec la fonction séparatrice. Par exemple, tous les points pour lesquels $f(\mathbf{x}) \geq 0$ sont étiquetés '+'. On obtient ainsi un ensemble d'apprentissage qui est utilisé par un algorithme d'apprentissage utilisant la programmation génétique. Cet algorithme utilise un jeu de terminaux incluant des constantes et des variables représentant les descripteurs (qui ne font pas tous forcément partie de la fonction cible: ce sont des descripteurs non pertinents) et un ensemble de fonctions incluant par exemple: $+$, $-$, \times ; \div , \cos , \sin , La fonction de performance mesure le risque empirique, par exemple par un critère entropique (voir les arbres de décision dans le chapitre 11), une fonction de coût quadratique ou simplement le nombre d'exemples mal classés. De plus, pour favoriser l'obtention d'expressions simples, un terme de pénalité a été introduit dans la fonction de performance, prenant en compte le nombre de nœuds dans les arbres produits. Une fois que l'algorithme a

trouvé une séparatrice, on évalue son risque réel en comptant le nombre d'exemples mal classés sur un échantillon de test.

Dans les expériences réalisées ([BTC96]), les paramètres avaient les valeurs suivantes :

- Taille de l'*ensemble d'apprentissage* : 200 exemples générés aléatoirement
- Taille de l'*ensemble de test* : 10000 exemples générés aléatoirement
- Taille de la *population* : 500 (i.e. 500 fonctions de décision en compétition dans la population)
- Nombre de *générations* : 50 (pour chaque exécution)
- Nombre d'*exécutions* : 1000
- *Profondeur maximale des arbres* : 8
- Ensemble de *fonctions* : $+, -, *, \div$ ($\cos, \sin, \exp, \log, \sqrt{\cdot}$)
- Taux des *opérateurs* :
 - reproduction: 10 %
 - croisement: 45 %
 - mutation: 45 %

Une exécution de l'algorithme peut ainsi produire une séquence de populations de fonctions séparatrices dont la meilleure évolue comme suit à différents instants :

$$\frac{\log(z)}{\frac{z}{y}} + x - \sin(z) \rightsquigarrow \frac{\frac{z}{y}}{\exp(x)} - \sqrt{\sin(z)} \rightsquigarrow x * \frac{z}{y} + \sqrt{x} \rightsquigarrow y * \frac{x}{z} \rightsquigarrow \frac{x \cdot y}{y+z}$$

8.5.5.2 Induction d'un programme de contrôle de robot simulé

Nous voulons simplement ici montrer comment un programme de contrôle simple en robotique peut être produit par programmation génétique. Le travail décrit succinctement dans cette section est tiré du livre de Christian Jacob *Illustrating evolutionary computation with Mathematica* [Jac01].

On s'intéresse à un robot placé dans un univers bidimensionnel consistant en une grille de 18×18 cases tel que celui de la figure 8.11. Quarante-quatre cases de nourriture ou de phéromones sont distribuées sur la grille. Ces cases sont organisées suivant un chemin avec d'abord des séquences de nourriture ininterrompues, puis de plus en plus de cases intermittentes de phéromones qui aident à orienter le robot fourmi. Le but du robot est d'essayer de visiter toutes les cases contenant de la nourriture en quatre-cents actions maximum. Les actions disponibles sont au nombre de quatre :

- **advance**: avancer d'une case en avant
- **turnLeft**: tourner de 90° à gauche
- **turnRight**: tourner de 90° à droite
- **nop**: pas d'action

Un robot peut également déterminer son action en fonction des conditions environnantes. Pour cela un robot est équipé d'un senseur qui fournit des informations sur la case qui se trouve directement devant : « mur », « nourriture », « phéromone » ou « poussière ». Cela autorise deux commandes conditionnelles :

- **ifSensor[signal] [A]** : l'action A est exécutée si le robot est devant une case correspondant à **signal**;
- **ifSensor[signal] [A,B]** : l'action A est exécutée si le robot est devant une case correspondant à **signal**, sinon c'est l'action B qui est exécutée.

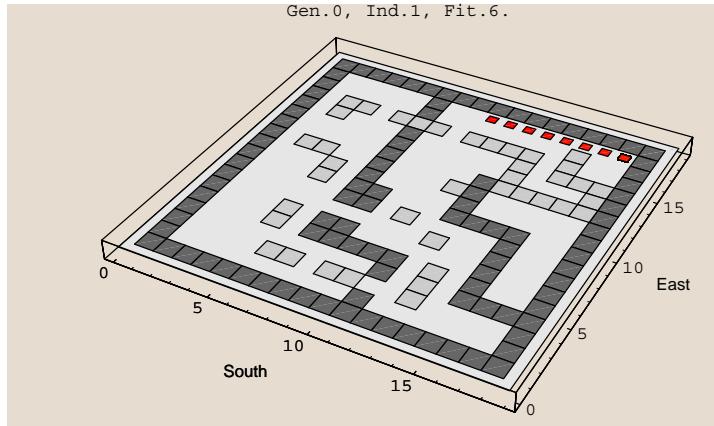


FIG. 8.11 – *Le monde du robot fourmi.* Les cases en noir correspondent à des murs infranchisables, celles en gris foncé à de la nourriture, celles en gris clair à des traces de phéromones. (D'après [Jac01], p.401).

Ces actions ou commandes sont intégrées dans une expression `seq` correspondant à une séquence. Chaque action ou commande, en comptant aussi les requêtes au senseur, est comptée comme une action. Le robot s'arrête s'il a trouvé toutes les cases contenant de la nourriture ou s'il a dépassé le nombre maximal d'actions autorisées, ici 400.

Un programme est représenté par une séquence de commandes ou d'actions qui peuvent éventuellement correspondre à des itérations si les commandes `stop` et `again` sont incluses.

Exemple de programmes

Par exemple, le programme dont l'expression est :

`ifSensor[food] [seq[advance, again], seq[nop]]`

correspond à une boucle permettant de suivre des cases consécutives de nourriture. La requête `ifSensor` sera exécutée aussi longtemps qu'il y a aura de la nourriture juste devant le robot déclenchant alors une action `advance`. Sinon, la boucle est quittée et le robot continuera d'exécuter les commandes suivantes s'il y en a.

La performance d'un programme de contrôle de robot h est mesurée par le nombre n de cases de nourriture visitées pondéré par un terme prenant en compte la complexité du programme, afin de favoriser les programmes simples :

$$\phi(h) = \begin{cases} n + 1 & \text{si } d_h \leq d_{max}, \\ \frac{d_{max}}{d_h} \cdot n + 1 & \text{si } d_h > d_{max}. \end{cases}$$

Les opérateurs génétiques utilisés dans les expériences décrites par Ch. Jacob incluent la mutation et le croisement, ainsi que des délétions, des duplications, des permutations, etc. Ces opérateurs sont définis par des moules (*templates*) qui spécifient quels motifs dans un programme peuvent être remplacés et par quoi. Par ailleurs, il existe un programme de simplification des programmes obtenus qui est appliqué à chaque génération. Ce type de programme de simplification est essentiel dans la programmation génétique.

Les figures 8.13 et 8.15 donnent une idée du genre de programmes qui peuvent être obtenus dans un processus d'évolution jouant sur une population d'individus de cinquante programmes. Le meilleur individu de la cinquième génération va droit devant avant d'être arrêté par le mur. À la génération 9 (voir la figure 8.12) le meilleur programme ne diffère du meilleur de la génération

5 que par une instruction, la dernière, mais celle-ci correspond à tourner à gauche si un mur est rencontré. La trajectoire obtenue à partir du point initial est alors une boucle. Des améliorations sensibles apparaissent de temps en temps lors des générations suivantes, jusqu'à l'obtention à la génération 59 d'un programme permettant, sur le problème posé, de passer sur toutes les cases de nourriture (voir figures 8.14 et 8.15).

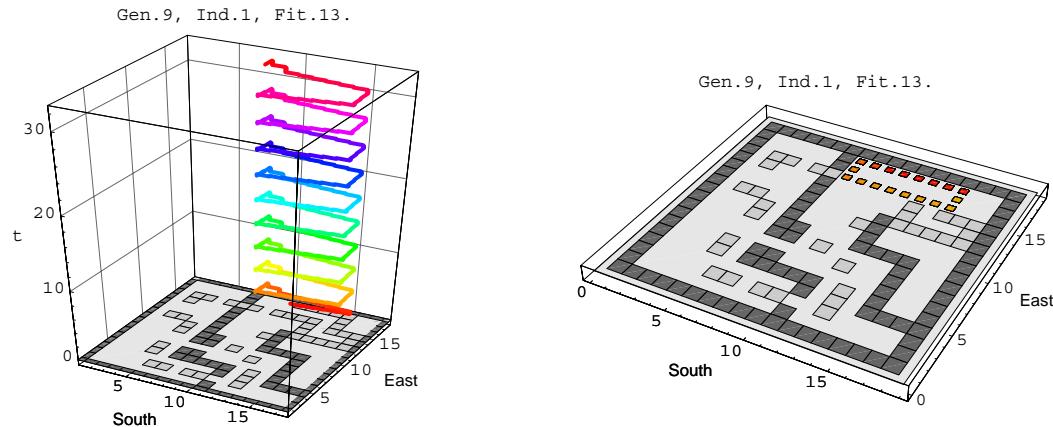


FIG. 8.12 – *Le meilleur individu de la génération 9 correspond à un robot qui avance tout droit sauf lorsqu'il se trouve face à un mur, auquel cas il tourne à gauche.* (D'après [Jac01], p.401).

```
AntTracker
[seq[advance, ifSensor[dust] [seq[stop], advance],
      ifSensor[dust] [seq[turnLeft, again], nop],
      advance,
      ifSensor[wall] [
        seq[turnLeft, again], advance]]]
```

FIG. 8.13 – *Le programme du meilleur individu de la génération 9.* (D'après [Jac01], p.401).

La figure 8.16 permet de visualiser l'évolution des performances des cinquante programmes sur les trente-neuf premières générations. On y observe une amélioration graduelle des performances moyennes avec des variations assez brutales de la performance du meilleur individu. La partie droite de cette figure montre l'évolution de la performance du meilleur individu, de la performance moyenne et de la performance du pire individu sur quatre-vingt-six générations.

Sur ces expériences, Ch. Jacob rapporte que sur vingt évolutions simulées, chacune limitée à cent générations, et impliquant une population de cinquante individus engendrés aléatoirement, dix programmes ont finalement été obtenus qui visitent les quarante-quatre cases de nourriture en moins de quatre-cents actions. Il serait évidemment intéressant de savoir comment se comportent ces programmes sur la même tâche mais dans un environnement différent.

8.5.6 Une analyse de la programmation génétique

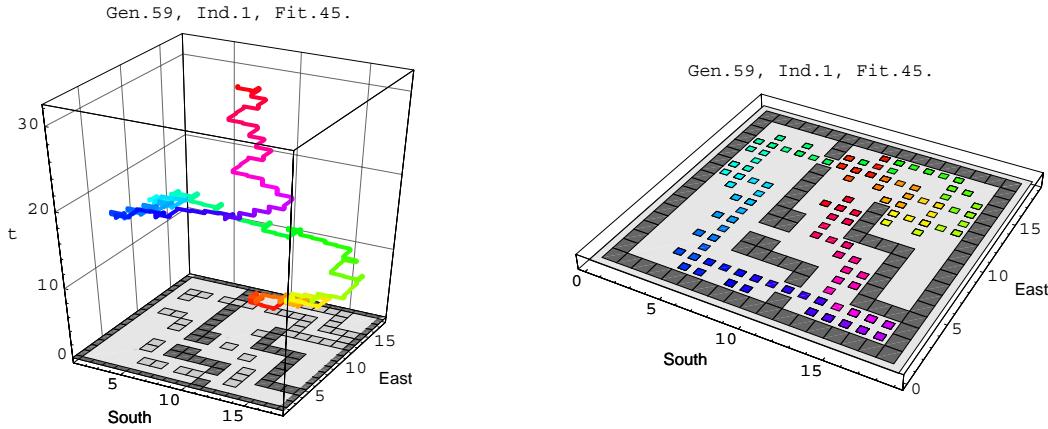


FIG. 8.14 – *Le comportement du meilleur programme de la génération 59. Toutes les cases de nourriture sont visitées sans détours inutiles. (D'après [Jac01], p.401).*

```
AntTracker
[seq[{},
  ifSensor[wall][seq[turnLeft, turnLeft, again],
    seq[{},
      ifSensor[wall][seq[turnLeft, again], turnRight],
      ifSensor[wall][seq[turnLeft, again], advance]]],
  ifSensor[dust][seq[turnLeft, again]],
  ifSensor[wall][seq[ifSensor[dust][advance]],
    seq[ifSensor[dust][seq[turnLeft, again]],
      ifSensor[wall][seq[turnLeft, again], advance]]]]]
```

FIG. 8.15 – *Le programme du meilleur individu de la génération 59. (D'après [Jac01], p.401).*

Le théorème des schémas est difficile à appliquer dans le cas de la programmation génétique notamment parce que les primitives de construction peuvent migrer d'un endroit à l'autre en cours d'évolution et aussi parce que la représentation des individus est de longueur variable. Nous reportons le lecteur intéressé à [BNKF98] pp.145 et suivantes pour une discussion sur les différentes tentatives d'analyse formelle de la programmation génétique. Il y a là encore un territoire à explorer.

Finalement, il est important de noter que dans le cas de la programmation génétique, les espaces des hypothèses \mathcal{H} et des génotypes \mathcal{G} sont de fait confondus. C'est en effet le même espace qui sert à l'exploration (rôle de \mathcal{G}) et à l'évaluation (rôle de \mathcal{H}). Un programme est ainsi à la fois l'objet subissant les opérateurs génétiques et l'objet s'exprimant dans l'univers pour y produire une certaine performance.

8.6 La coévolution

Les approches mentionnées jusqu'ici cherchent à optimiser une fonction de performance fixée une fois pour toutes par l'objectif et par l'environnement connu par l'intermédiaire d'un

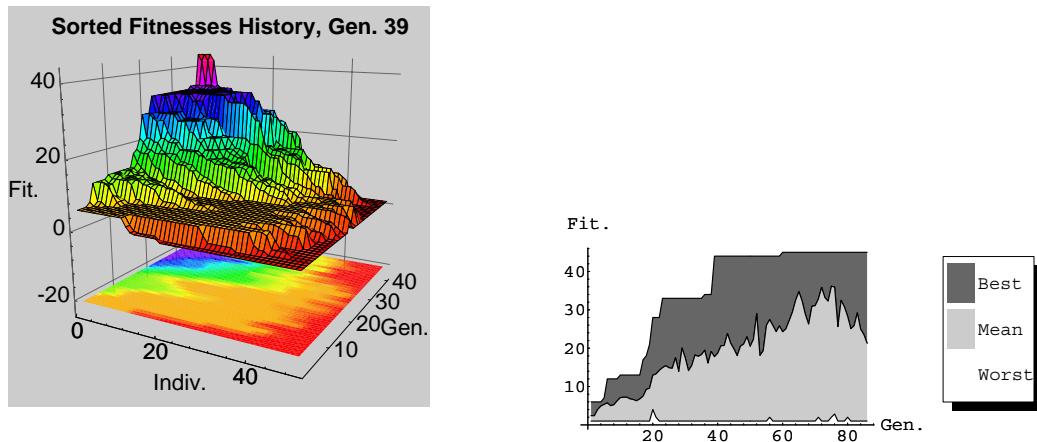


FIG. 8.16 – À gauche, l'évolution des performances des cinquante programmes sur les trente-neuf premières générations. À droite, l'évolution des performances du meilleur individu, de la moyenne et du pire individu, sur quatre-vingt-cinq générations. (D'après [Jac01], p.418).

échantillon d'apprentissage. La recherche d'une bonne hypothèse s'effectue alors dans l'espace des hypothèses \mathcal{H} grâce à des manipulations dans l'espace génotypique \mathcal{G} . On espère que la représentation choisie pour \mathcal{G} et les opérateurs de transformation sélectionnés rendront la recherche efficace. Cependant, il est peut-être possible de faire mieux encore en *adaptant dynamiquement la fonction de performance* que l'algorithme cherche à optimiser. L'idée serait de faciliter l'exploration de l'espace d'hypothèses en cherchant d'abord des optima faciles à trouver, puis en raffinant progressivement le critère objectif. D'une certaine manière, il s'agit là de l'idée mise en œuvre dans le recuit simulé par exemple. Un processus de recherche de moins en moins stochastique permettait de focaliser le système sur les zones les plus prometteuses de l'espace de recherche. Dans le cas des algorithmes opérant sur le génotype par évolution de populations, il est possible de réaliser cette idée d'une autre manière inspirée par l'observation de l'évolution dans la nature.

Le principe est de faire dépendre la fonction de performance utilisée pour la sélection des meilleurs individus d'une population $P_{\mathcal{G}_1}(t)$ d'une autre population $P_{\mathcal{G}_2}(t)$, elle-même sous la dépendance d'une fonction de performance commandée par $P_{\mathcal{G}_1}(t)$. La métaphore biologique est celle de la coévolution proie-prédateur, dans laquelle au fur et à mesure que les prédateurs améliorent leur performance contre les proies, celles-ci modifient leur comportement pour diminuer leur vulnérabilité. Il s'ensuit une sorte de course aux armements qui à la fois rend plus performants les individus des deux populations, mais aussi les rend davantage spécialisés face à un environnement.

D'un point de vue pratique, plusieurs scénarios permettent de réaliser une fonction de performance dépendant d'une autre population. Par exemple, dans certains cas, il est possible de tester chaque individu contre tous les individus de la population adverse. On peut aussi tester chaque individu contre le meilleur adversaire. Ou encore, on peut tester des individus pris au hasard dans une population contre des adversaires également tirés au hasard dans l'autre population. De la sorte, chaque population devrait améliorer sa performance face à la fonction de

performance évoluant dynamiquement.

Cette approche a été étudiée dans le cadre de la modélisation de phénomènes naturels. Mais elle a aussi été utilisée pour des problèmes d'optimisation. Ainsi, [Hil92, AP93, Sim94] ont documenté des améliorations parfois considérables obtenus par coévolution dans plusieurs problèmes d'optimisation. Par exemple, [Ron96] a étudié le problème du robot fourmi décrit plus haut dans lequel les morceaux de nourriture eux-mêmes évoluaient afin déchapper à une population de robots-fourmi. Il a montré alors une évolution plus rapide du comportement des robots-fourmi. Cette approche a également été utilisée par Jannink ([Jan94]) pour optimiser des générateurs de nombres pseudo-aléatoires. L'idée était qu'une population de testeurs de caractère aléatoire de séquences produites par une population de générateurs coévoluait avec celle-ci.

L'idée de coévolution est encore préliminaire et ouverte à recherche. Une voie à explorer est celle de coopération plutôt que celle de confrontation. La théorie des jeux est un guide utile en ce domaine.

Citons pour finir les travaux sur l'évolution des opérateurs génétiques eux-mêmes pendant le processus d'optimisation.

8.6.1 Un exemple d'écologie : les systèmes de classeurs

Un intéressant exemple d'écologie d'hypothèses est fourni par le systèmes de classeurs (*classifier systems*) imaginé par John Holland en 1975, bien avant que l'on parle de coévolution.

Le principe est celui d'un agent placé dans un environnement dynamique, contrôlé par un système expert dont les règles servent à interpréter le monde et à prendre des décisions qui sont alors mises en œuvre à l'aide d'effecteurs. Les règles appelées *classeurs* ont le format suivant : **if condition then action**.

Comme dans un système expert classique, chaque règle peut être déclenchée par la situation courante qui résulte à la fois de la perception et des inférences conduites par le système. Cette situation est représentée dans une mémoire à court terme ou mémoire de travail. À chaque instant l'ensemble des règles courantes (la population) examine la mémoire de travail pour voir si ses conditions de déclenchement sont réalisées. Contrairement à un système expert classique où seule une règle est déclenchée après sélection, ici toutes les règles déclenchables sont déclenchées et contribuent ainsi à l'inférence courante. Celle-ci modifie alors la situation courante et le cycle recommence (voir figure 8.17).

D'un certain côté, on peut concevoir ce système de règles comme une écologie avec des règles se spécialisant dans certains types de tâches et contribuant ainsi à la performance globale. Le problème est de faire évoluer un tel système écologique. Pour ce faire, Holland a proposé un système de rétribution de mérite fonctionnant un peu comme une économie où des entreprises en compétition pour des contrats rémunèrent leurs sous-traitants, devenant plus riches si le contrat est remporté, ou s'appauvrissant dans le cas contraire. Cette redistribution des mérites est assez proche des méthodes d'apprentissage par renforcement avec trace d'éligibilité (voir chapitre 16). Le mérite attribué de la sorte sert alors de mesure de performance pour chaque règle et le fonctionnement classique d'un algorithme génétique est alors utilisé pour déterminer les parents de la génération suivante.

Nous n'entrons pas davantage dans les détails, reportant le lecteur intéressé à [BGH89]. Malgré la séduction exercée par cette idée, elle n'a pas permis jusqu'à présent d'obtenir de résultats probants, notamment parce que la redistribution des mérites est très lente et ne permet pas une évolution rapide de la population. Peut-être que les prochaines générations de machines et des conceptions neuves sur la coévolution donneront un jour une nouvelle jeunesse à cette idée.

séduisante car elle correspond à l'évolution d'une société d'individus spécialisés fonctionnant sur le mode de compétition/coopération.

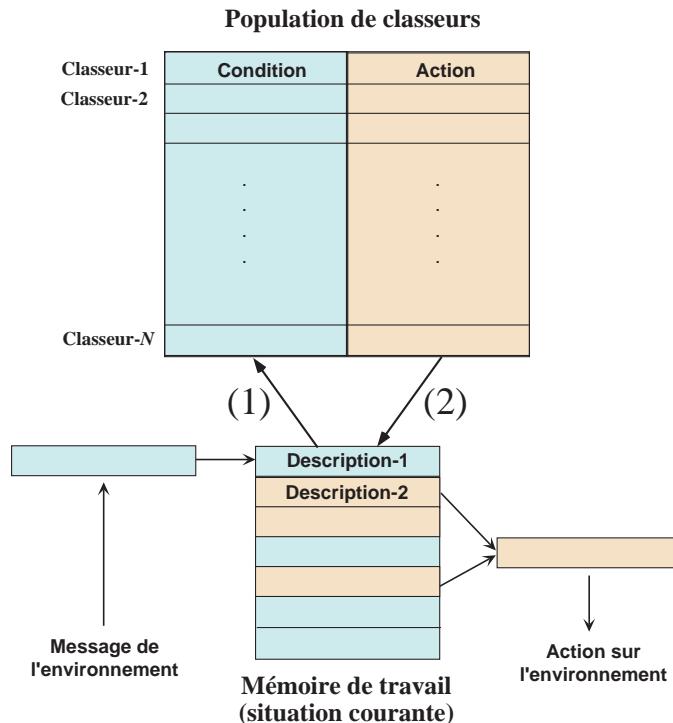


FIG. 8.17 – Schéma d'un système de classeurs. En (1), les messages de la mémoire de travail sont comparés aux règles du système. En (2), les règles déclenchées inscrivent des messages dans la mémoire de travail, correspondant soit à des interprétations sur le monde, soit à des décisions qui seront mises en œuvre par des effecteurs.

Notes historiques et approfondissements

La métaphore de l'évolution comme base possible pour des algorithmes d'apprentissage en intelligence artificielle remonte aux années 1950 et 1960. Dans les années 1960, Rechenberg introduisit l'approche des *stratégies d'évolution* qu'il utilisa pour optimiser des fonctions à paramètres à valeurs réelles pour la conception de systèmes mécaniques. Cette idée a été développée par Schwefel en Allemagne dans les années 1970. Elle connaît un grand regain d'intérêt ces dernières années car elle s'applique assez naturellement à de nombreux problèmes d'optimisation. Par ailleurs, elle évite certains problèmes de codage que l'on rencontre avec les algorithmes basés sur une représentation par chaînes de bits. Nous n'en avons pas parlé par manque de place, mais cette approche a fait l'objet récemment de nombreux articles dans des conférences et des revues portant sur les algorithmes d'évolution simulée.

Les premiers modèles de processus d'adaptation portant sur des populations d'individus par *algorithmes génétiques* sont dus à John Holland et ont été popularisés par son ouvrage [Hol75]. Outre une modélisation de l'adaptation par utilisation d'opérateurs génétiques et de sélection des meilleurs individus d'une population à la suivante, Holland proposa une analyse théorique de ces algorithmes par le théorème des schémas. Son intérêt principal alors et maintenant portait sur l'évolution de population de règles (*classifier systems*). Les algorithmes génétiques connurent

un vif développement dans les années 1980 et lancèrent véritablement le mouvement pour les algorithmes évolutionnaires.

L'approche initiée par Fogel, Owens et Walsh dans les années 1960 sur les *stratégies d'évolution* dans le cadre de populations d'automates à états finis reste minoritaire dans les recherches et les applications actuelles. Il n'est pas impossible que le développement des applications en prédition temporelle ne relance cette voie.

Une approche liée aux stratégies d'évolution et qui étend les algorithmes génétiques de Holland à des représentations plus larges est celle de la *programmation génétique* dont Koza s'est fait un avocat puissant [Koz92, Koz94, KA99]. Des conférences et des revues sont maintenant entièrement consacrées à cette famille d'algorithmes.

Globalement, la technique des algorithmes d'évolution artificielle est devenue très populaire, autant que les réseaux connexionnistes. Elle le doit en particulier au fait qu'elle peut s'appliquer dans de très nombreux contextes pour lesquels les techniques d'optimisation traditionnelles sont mises en échec par manque de « régularité » de l'espace des fonctions cible. Les recherches sont actives, tant pour investiguer de nouvelles variations algorithmiques que pour essayer d'en faire une analyse théorique. Cette dernière est difficile. Elle est liée à la fois aux théories de l'optimisation et à des théories sur les processus de diffusion par exemple.

Résumé

Les algorithmes évolutionnaires exploitent un découplage de l'espace d'hypothèses en un *espace phénotypique* (dans lequel les hypothèses ont une description portant sur le monde et sont donc évaluables) et un *espace génotypique* (dans lequel les hypothèses, représentées par des génomes, subissent un processus d'évolution simulée). L'espoir est que les *opérateurs génétiques* utilisés pour faire évoluer les hypothèses, en particulier le croisement et la mutation, fassent évoluer la population d'hypothèses vers les régions de \mathcal{H} où se trouvent les hypothèses les meilleures.

L'un des intérêts des algorithmes évolutionnaires est leur large domaine d'applications puisqu'ils permettent d'explorer des espaces d'hypothèses dans lesquels la fonction de performance attachée aux hypothèses n'est pas dérivable. C'est pourquoi ils sont fréquemment utilisés.

Plusieurs familles d'algorithmes ont été développées, dont principalement : les *algorithmes génétiques* utilisant plutôt une représentation binaire des génomes, les *stratégies d'évolution* reposant sur une représentation par automates, la *programmation génétique* travaillant dans un espace de programmes représentable par des arbres. Il existe également une extension appelée *coévolution* pour laquelle la fonction d'évaluation des hypothèses elle-même résulte d'un processus évolutif dans un jeu de compétition.

Troisième partie

Apprentissage par optimisation

Chapitre 9

L'apprentissage de surfaces séparatrices linéaires

Ce chapitre explique comment on peut apprendre un concept sous la forme géométrique la plus simple : celle d'un hyperplan. En pratique, on dispose d'exemples et de contre-exemples dans l'espace de représentation, qui est ici nécessairement numérique. On cherche à trouver une formule aussi simple que possible pour généraliser ces éléments d'apprentissage. Cette formule est une combinaison linéaire sur les attributs, ayant la propriété suivante : quand un objet inconnu est à classer comme appartenant au concept ou rejeté par le concept, on effectue un calcul linéaire sur les attributs de cet objet. Le signe du résultat indique la décision à prendre. Géométriquement, cela revient à dire qu'une fois terminée la phase d'apprentissage, l'espace de représentation est partagé en deux par une surface linéaire et que l'un des demi-espaces correspond au concept, l'autre à sa négation.

Ces méthodes ont une longue histoire en informatique, puisqu'on les a proposées comme l'archétype de l'apprentissage artificiel dès le début des années 1960. Elles ont toujours une grande importance, pour plusieurs raisons. La première est la simplicité du calcul de la décision. Ensuite, elles sont robustes au bruit. D'autre part, on dispose d'une variété d'algorithmes d'apprentissage, dont la plupart permettent une interprétation sous la forme de l'optimisation d'un critère compréhensible sur l'ensemble d'apprentissage. Ces méthodes ont été généralisées à la fin des années 1980 par les réseaux connexionnistes : il est difficile de bien comprendre le fonctionnement de ceux-ci sans connaître leur version primitive. Enfin, dans les années 1990, une nouvelle impulsion a été donnée à ce type de méthodes par l'invention des efficaces SVM (support vector machines ou séparateurs à vaste marge), qui font subir une transformation non linéaire à l'espace de représentation avant d'y chercher une surface séparatrice linéaire.

DANS L'AVANT-PROPOS, nous avons donné un exemple d'apprentissage d'une règle de décision sous la forme d'une droite dans l'espace de représentation : on se souvient que les cygnes et les oies étaient représentés par deux attributs observables, leur niveau de gris et leur taille. La formule magique permettant de décider à laquelle de ces deux espèces appartenait un oiseau inconnu était le calcul d'une combinaison linéaire sur les valeurs numériques de ces deux attributs. Ce chapitre étudie de manière plus formelle les caractéristiques de cette formule magique, sa mise en œuvre et ses propriétés.

Nous allons reprendre l'exemple des cygnes de manière un peu différente pour introduire les problématiques qui seront développées dans ce chapitre. D'abord, donnons un ensemble d'apprentissage, sur les mêmes deux dimensions, composé d'observations d'oies (notées ici O) et de cygnes (Z).

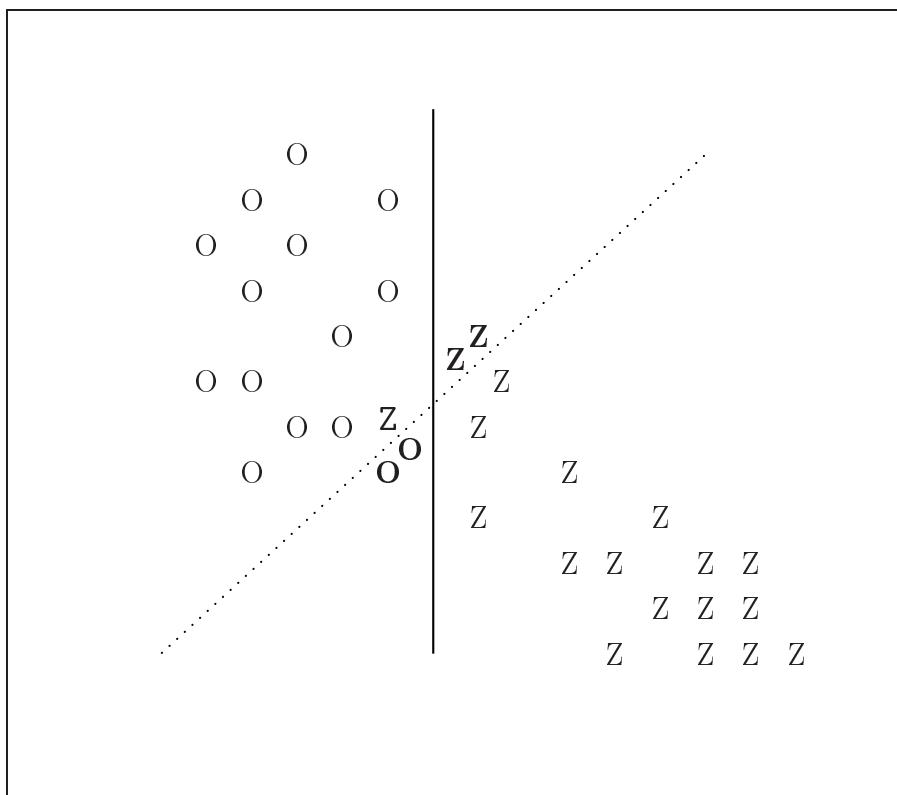


FIG. 9.1 – Deux séparations linéaires possibles pour deux classes.

Pour trouver le concept géométrique qui définit la différence entre un cygne et une oie, il suffit de tracer une droite dans cet espace de représentation. Comment tirer au mieux parti des données d'apprentissage ? Considérons deux critères simples. Le premier répond au principe d'apprentissage *ERM* : on cherche la droite qui minimise le nombre d'erreurs dans l'ensemble d'apprentissage. Elle est donnée en trait plein sur la figure. On remarque qu'un seul oiseau est mal classé par l'hyperplan séparateur, qui est ici la droite verticale. Ce point est indiqué par le graphisme **Z**.

L'autre critère représenté sur la figure est un peu plus élaboré : il minimise une quantité qui tient compte de tous les points d'apprentissage et répond au principe bayésien : la droite obtenue, tracée en pointillés, bien qu'elle ait une erreur apparente de cinq éléments d'apprentissage (indiqués en gras), est plus équilibrée et reflète mieux la géométrie globale du concept à

apprendre. Nous ne serons pas plus précis pour le moment : la formalisation de ce type de critère sera donnée dans ce chapitre. Constatons simplement que si on suppose, comme il se doit, que les points d'apprentissage sont représentatifs de la géométrie du concept cherché, cette seconde droite semble mieux prédire la répartition des observations dans l'espace choisi.

Notations utiles pour le chapitre

$\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix}$	Un vecteur
$\mathbf{x}^T = (x_1, \dots, x_d)$	Un vecteur transposé
$\ \mathbf{x}\ $	La norme du vecteur \mathbf{x}
M^{-1}	La matrice inverse d'une matrice carrée M
M^T	La matrice transposée d'une matrice M
M^+	La matrice pseudo-inverse d'une matrice M . Par définition, $M^+ = M^T(MM^T)^{-1}$
$\Delta(\mathbf{x}, \mathbf{y})$	La distance euclidienne entre deux vecteurs \mathbf{x} et \mathbf{y} de \mathbb{R}^d
$\frac{\partial}{\partial x} f(x, y)$	La dérivée partielle par rapport à x de la fonction f des deux variables x et y
$\nabla_{\mathbf{A}} J(\mathbf{A}, \mathbf{B})$	Le vecteur dérivé par rapport au vecteur \mathbf{A} de la fonctionnelle J des deux vecteurs \mathbf{A} et \mathbf{B}

9.1 Généralités.

9.1.1 Hyperplans séparateurs et discriminants dans un problème à deux classes.

Quand on est dans un espace de représentation euclidien, on peut librement faire des hypothèses sur la géométrie des classes ou sur celles de leurs surfaces séparatrices ; ceci permet de mettre au point des techniques d'apprentissage non statistiquement fondées *a priori*, mais peut-être plus faciles à appliquer. La plus simple d'entre elles est de supposer que deux classes peuvent être séparées par une certaine surface, définie par une équation ; les paramètres qui régissent cette équation sont alors les variables à apprendre. Cette hypothèse n'est pas au fond tellement plus forte que de supposer que les classes sont de nature gaussienne, comme on le fera par exemple au chapitre 14 ; elle est souvent bien sûr fausse, mais peut mener à une probabilité d'erreur raisonnable dans un bon nombre de problèmes. Cette probabilité d'erreur doit évidemment être évaluée objectivement, sur un ensemble de test ou par les méthodes présentées au paragraphe 3.4.5.1.

Le nombre de paramètres à calculer est minimal si l'on suppose cette surface linéaire ; aussi est-ce l'hypothèse qui prévaut dans la plupart des cas, d'autant qu'elle permet de mener des calculs faciles et de visualiser précisément le résultat obtenu.

Dans \mathbb{R}^d , une surface linéaire est un hyperplan \mathbf{A} , défini par l'équation :

$$a_0 + \mathbf{a}^T \mathbf{x} = 0 \quad (9.1)$$

avec \mathbf{a} vecteur de dimension d et a_0 scalaire. Si deux classes ω_1 et ω_2 sont *séparables* par \mathbf{A} , tous les points de la première classe sont par exemple tels que :

$$\mathbf{x} \in \omega_1 \Rightarrow a_0 + \mathbf{a}^T \mathbf{x} \geq 0 \quad (9.2)$$

et ceux de la seconde vérifient alors :

$$\mathbf{x} \in \omega_2 \Rightarrow a_0 + \mathbf{a}^T \mathbf{x} \leq 0 \quad (9.3)$$

Dans un espace de dimension $d = 1$, une séparation linéaire se réduit à la comparaison à un seuil. Prenons ce cas particulier pour donner deux exemples où un problème de discrimination à deux classes ne peut pas en pratique être complètement résolu par une séparatrice linéaire.

Exemple 4

Cherchons à classer les hommes et les femmes sur la seule mesure de leur taille. Bien que les hommes soient en général plus grands que les femmes, aucun échantillon d'apprentissage représentatif ne permettra de fixer une valeur pertinente pour le seuil, étant donné qu'il existera évidemment toujours certains hommes plus petits que certaines femmes. Dans ce cas, c'est l'espace de représentation qui est trop pauvre: d'autres critères doivent être mis en œuvre pour décider sans erreur du sexe d'un individu.

Supposons pour un autre exemple qu'un avimateur cherche à discriminer les eiders (*Somateria mollissima*) mâles des femelles par le niveau de gris moyen de leur plumage. Là encore, un seuil sur cette mesure donnera une classification confuse; mais, cette fois, l'espace de mesure est suffisant; c'est en revanche la définition des classes qui est trop pauvre. En effet, les femelles sont toutes marron clair; mais les mâles sont soit blanc brillant quand on les observe de face, soit noirs quand ils sont vus de dos. Il faudrait donc subdiviser cette dernière espèce, ce qui rendrait possible une bonne séparation linéaire en trois classes (eider mâle vu de face, eider mâle vu de dos, eider femelle) par deux seuils sur les couleurs. Il suffirait ensuite de regrouper les deux classes extrêmes pour avoir résolu le problème dans l'espace de représentation initial. Cet exemple sera détaillé au début du chapitre suivant, pour présenter les réseaux connexionnistes.

Définition 9.1 (Hyperplan séparateur. Séparatrice linéaire)

On appelle hyperplan séparateur ou séparatrice linéaire un hyperplan qui sépare parfaitement les deux classes, c'est-à-dire qui vérifie les équations 9.2 et 9.3; en particulier, il sépare parfaitement leurs points d'apprentissage.

Il n'est en général pas possible d'en trouver un. On se contentera donc de chercher un hyperplan discriminant qui en sera une approximation, au sens d'un critère à fixer.

Le problème de l'apprentissage de surfaces linéaires n'est autre que la recherche des paramètres \mathbf{a} et a_0 séparant le mieux possible les points d'apprentissage de ω_1 et de ω_2 dans l'espace \mathbb{R}^d et pourvues de la meilleure faculté de généralisation possible.

9.1.2 Un peu de géométrie dans \mathbb{R}^d

Un hyperplan dans \mathbb{R}^d , où les coordonnées sont notées x_1, \dots, x_d , est défini par $d + 1$ paramètres a_0, a_1, \dots, a_d . Il répond à l'équation :

$$a_0 + a_1 x_1 + \dots + a_d x_d = 0 \quad (9.4)$$

En notant vectoriellement $\mathbf{a}^T = (a_1, \dots, a_d)$ et $\mathbf{x}^T = (x_1, \dots, x_d)$, l'équation 9.4 s'écrit de manière équivalente: $a_0 + \mathbf{a}^T \mathbf{x} = 0$.

Un hyperplan d'équation $a_0 + \mathbf{a}^T \mathbf{x} = 0$ a pour vecteur normal \mathbf{a} . Sa distance algébrique à l'origine vaut $\frac{a_0}{\|\mathbf{a}\|}$ avec $\|\mathbf{a}\|^2 = \sum_{i=1}^d a_i^2$.

La figure 9.2 montre un hyperplan à deux dimensions qui est la droite d'équation $g(\mathbf{x}) = a_0 + a_1 x_1 + a_2 x_2 = 0$, ainsi que son vecteur normal \mathbf{a} .

Cette figure donne aussi la distance de deux points particuliers à cet hyperplan: l'origine, à la distance $\frac{a_0}{\|\mathbf{a}\|}$, et un point \mathbf{y} de l'autre côté de la droite, à la distance $\frac{g(\mathbf{y})}{\|\mathbf{a}\|}$. Les signes de ces deux distances algébriques sont opposés, puisque les deux points ne sont pas du même côté de l'hyperplan. Le signe de la distance de l'origine à l'hyperplan est le même que celui de a_0 .

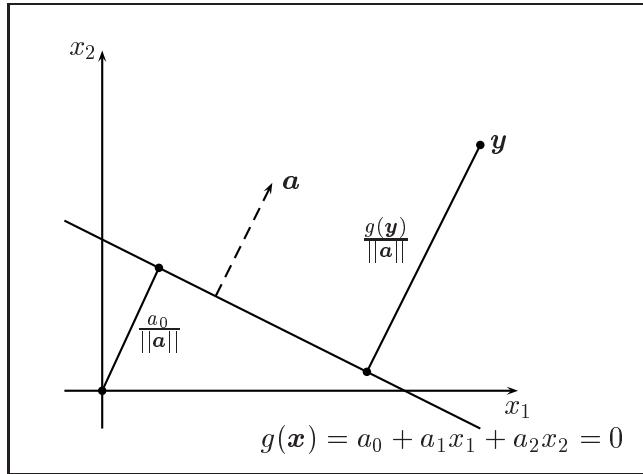


FIG. 9.2 – La géométrie d'un hyperplan.

Pour caractériser l'hyperplan d'équation $a_0 + \mathbf{a}^T \mathbf{x} = 0$, on peut rassembler tous ses paramètres dans un vecteur \mathbf{A} de dimension $d + 1$ construit comme :

$$\mathbf{A}^T = (a_0, \mathbf{a}^T) = (a_0, a_1, \dots, a_d)$$

Par extension, on note \mathbf{A} l'hyperplan lui-même. Notons qu'en réalité il suffit de d paramètres pour définir un hyperplan : on peut sans perdre de généralité imposer par exemple : $a_0 = 1$.

Posons $g(\mathbf{x}) = a_0 + \mathbf{a}^T \mathbf{x}$ et notons $\Delta(\mathbf{y}, \mathbf{A})$ la distance entre l'hyperplan \mathbf{A} d'équation $g(\mathbf{x}) = 0$ et un point \mathbf{y} de \mathbb{R}^d . Elle est définie comme la distance entre \mathbf{y} et le point de \mathbf{A} le plus proche de \mathbf{y} . Ce point n'est autre que la projection orthogonale de \mathbf{y} sur \mathbf{A} . La valeur $\Delta(\mathbf{y}, \mathbf{A})$ est, comme on l'a vu, une distance algébrique donnée par la formule :

$$\Delta(\mathbf{y}, \mathbf{A}) = \frac{g(\mathbf{y})}{\|\mathbf{a}\|}$$

9.2 L'apprentissage d'un hyperplan pour discriminer deux classes

9.2.1 Une solution globale

Supposons pour le moment que les deux classes ω_1 et ω_2 soient parfaitement séparables par un hyperplan. Il existe donc un vecteur \mathbf{a} caractérisant cet hyperplan séparateur tel que :

- $\mathbf{a}^T \mathbf{x}$ positif pour $\mathbf{x} \in \omega_1$;
- $\mathbf{a}^T \mathbf{x}$ négatif pour $\mathbf{x} \in \omega_2$.

Notons qu'il est facile de se ramener à ce cas quand c'est le contraire qui se produit, en inversant simplement le signe de toutes les coordonnées de \mathbf{a} . La formalisation de la séparation linéaire pour un problème à deux classes se fait alors en plongeant le problème dans l'espace de dimension $d + 1$.

Le vecteur \mathbf{a} et le scalaire a_0 sont transformés en un vecteur \mathbf{A} en ajoutant a_0 à \mathbf{a} comme une coordonnée de rang 0, ce qui se note: $\mathbf{A}^T = (a_0, \mathbf{a}^T)$. On impose une valeur non nulle à a_0 , par exemple $a_0 = 1$.

Soit \mathcal{S} l'ensemble des données d'apprentissage: il est composé de m exemples (\mathbf{x}, ω) où \mathbf{x} est un vecteur de \mathbb{R}^d et $\omega \in \{\omega_1, \omega_2\}$. Au besoin, on notera \mathbf{x}_j le $j^{ème}$ point de A . On transforme ces exemples en vecteurs \mathbf{X} de \mathbb{R}^{d+1} par :

$$\mathbf{x} \in \omega_1 \Rightarrow \mathbf{X}^T = (1, \mathbf{x}^T) \quad (9.5)$$

$$\mathbf{x} \in \omega_2 \Rightarrow \mathbf{X}^T = (-1, -\mathbf{x}^T) \quad (9.6)$$

Par conséquent, quelle que soit la classe de \mathbf{x} , on a désormais: $\mathbf{A}^T \mathbf{X} \geq 0$ (rappelons que l'hyperplan caractérisé par le vecteur \mathbf{A} est pour le moment supposé séparateur: tous les points sont bien classés).

Si on range maintenant les m vecteurs \mathbf{X} comme les colonnes d'une matrice M , il est alors facile de voir le problème de séparation linéaire comme la recherche d'un vecteur \mathbf{A} dans \mathbb{R}^{d+1} tel que:

$$\mathbf{A}^T M = \mathbf{B}^T \quad (9.7)$$

où \mathbf{B} est un *vecteur positif* de \mathbb{R}^m (dont toutes les coordonnées sont positives).

Ceci revient à dire que l'on recherche un hyperplan dans l'espace de représentation augmenté d'une dimension, avec les contraintes qu'il passe par l'origine et que \mathbf{B} reste positif. Cette manipulation mathématique n'a d'autre but que de rendre plus aisée la résolution du problème.

Cette équation ne peut pas se résoudre dans le cas général, puisque M n'est pas inversible (c'est une matrice à $d+1$ lignes et m colonnes) et que \mathbf{B} est inconnu. De plus, il faut maintenant traiter le cas général où les classes et les données d'apprentissage ne sont pas séparables. On se ramène donc à la recherche d'une séparatrice linéaire en cherchant un hyperplan \mathbf{A} le meilleur possible pour distinguer les deux classes. Par exemple, celui qui minimise le critère :

$$J(\mathbf{A}, \mathbf{B}) = \frac{1}{2} \| \mathbf{A}^T M - \mathbf{B}^T \|_F^2 \quad (9.8)$$

Pourquoi choisir cette valeur? Si les deux classes sont séparables, il existera un couple (\mathbf{A}, \mathbf{B}) avec \mathbf{B} positif pour lequel ce critère sera exactement nul. En imposant dans le cas général la contrainte \mathbf{B} positif ou nul, on peut donc comprendre intuitivement qu'une bonne solution approchée pour \mathbf{A} rendra $J(\mathbf{A}, \mathbf{B})$ assez petit. Donnons une justification plus profonde à cet argument.

$J(\mathbf{A}, \mathbf{B})$ peut s'écrire :

$$J(\mathbf{A}, \mathbf{B}) = \frac{1}{2} \sum_{j=1}^m [\mathbf{A}^T \mathbf{X}_j - b_j]^2$$

\mathbf{X}_j étant la projection dans l'espace de dimension $d + 1$ du $j^{ème}$ point d'apprentissage \mathbf{x}_j . On sait (voir le chapitre 3, paragraphe 9.1.2) que la distance $\Delta(\mathbf{x}_j, \mathbf{A})$ du point \mathbf{x}_j à l'hyperplan caractérisé par le vecteur \mathbf{A} a pour valeur :

$$\Delta(\mathbf{x}_j, \mathbf{A}) = \frac{\mathbf{a}^T \mathbf{x}_j + a_0}{\| \mathbf{a} \|} = \frac{\mathbf{A}^T \mathbf{X}_j}{\| \mathbf{a} \|}$$

Son carré peut donc s'écrire :

$$[\Delta(\mathbf{x}_j, \mathbf{A})]^2 = \frac{(\mathbf{A}^T \mathbf{X}_j)^2}{\|\mathbf{a}\|^2}$$

On en déduit :

$$J(\mathbf{A}, \mathbf{B}) = \frac{1}{2\|\mathbf{a}\|^2} \sum_{j=1}^m [\Delta(\mathbf{x}_j, \mathbf{A}) - b_j \|\mathbf{a}\|]^2$$

Pour les points de \mathbf{A} bien classés par l'hyperplan \mathbf{A} , le terme $[\Delta(\mathbf{x}_j, \mathbf{A}) - b_j \|\mathbf{a}\|]^2$ peut être annulé en choisissant $b_j = \frac{\Delta(\mathbf{x}_j, \mathbf{A})}{\|\mathbf{a}\|}$, qui est positif. Pour les points mal classés, à défaut de pouvoir prendre b_j négatif, le mieux que l'on puisse faire est d'imposer b_j nul.

Minimiser $J(\mathbf{A}, \mathbf{B})$ sous la contrainte \mathbf{B} positif ou nul revient donc d'une certaine manière à se placer dans le cadre du critère des moindres carrés 3 : on cherche l'hyperplan \mathbf{A} qui minimise la somme des distances entre \mathbf{A} et les points de l'ensemble d'apprentissage que \mathbf{A} classe mal. On a, si \mathbf{A} et \mathbf{B} minimisent effectivement $J(\mathbf{A}, \mathbf{B})$:

$$[J(\mathbf{A}, \mathbf{B})] = \sum_{\mathbf{x} \in \mathcal{S} \text{ mal classé par } \mathbf{A}} [\Delta(\mathbf{x}, \mathbf{A})]^2$$

Par conséquent, sous l'hypothèse que les distances entre \mathbf{A} et les points mal classés par \mathbf{A} sont réparties selon une distribution gaussienne, et que ceux-ci sont les seuls à compter dans le positionnement de \mathbf{A} , la minimisation de $J(\mathbf{A}, \mathbf{B})$ est la recherche du meilleur hyperplan *a posteriori* (au sens bayésien), c'est-à-dire le plus probable connaissant les données (voir le chapitre 14, paragraphe 14.1.6).

Résolution

Les équations classiques de l'algèbre linéaire fournissent la valeur du gradient de $J(\mathbf{A}, \mathbf{B})$ par rapport à \mathbf{A} :

$$\nabla_{\mathbf{A}} J(\mathbf{A}, \mathbf{B}) = (\mathbf{A}^T M - \mathbf{B}^T) M^T \quad (9.9)$$

Comme $J(\mathbf{A}, \mathbf{B})$ est toujours positif ou nul, son minimum est atteint pour :

$$\nabla_{\mathbf{A}} J(\mathbf{A}, \mathbf{B}) = 0 \quad (9.10)$$

ce qui fournit quand la matrice $M^T M$ est inversible (ce qui est vrai en général) :

$$\mathbf{A}^T = \mathbf{B}^T M^+ \quad (9.11)$$

avec $M^+ = M^T (M M^T)^{-1}$ *pseudo-inverse* de M .

Par conséquent, si le vecteur \mathbf{B} était connu, la solution à notre problème consisterait simplement à calculer \mathbf{A} par la formule :

$$\mathbf{A}^T = \mathbf{B}^T M^+ \quad (9.12)$$

À défaut de connaître exactement \mathbf{B} , on sait qu'il est positif quand les données sont séparables. On peut, conformément à ce qui a été dit plus haut, chercher une solution qui approche ce cas idéal en fournissant des valeurs strictement positives pour les coordonnées de \mathbf{B} correspondant à des points bien classés, et des valeurs nulles pour les points mal classés. Dans cette optique, l'algorithme cherché doit réaliser une minimisation de $J(\mathbf{A}, \mathbf{B})$ sous la contrainte \mathbf{B} positif ou nul.

On dispose alors de deux types de méthodes pour finir de résoudre le problème: celles reposant sur un calcul numérique global direct et celles qui procèdent de manière itérative sur les données d'apprentissage (on en verra une au paragraphe suivant). La méthode globale la plus simple est la *programmation linéaire*: si l'on ne retient de \mathbf{B} que sa positivité, l'équation ci-dessus se ramène à m inéquations linéaires dans l'espace de dimension $d + 1$. Le problème est donc de grande taille si le nombre d'exemples est important. Les méthodes globales sont donc peu employées ici.

9.2.2 Une méthode itérative: l'algorithme de Ho et Kashyap.

Cette méthode consiste à partir d'un vecteur \mathbf{B}_0 arbitraire, puis à en déduire une suite convergente de valeurs $\mathbf{A}_{(t)}$ et $\mathbf{B}_{(t)}$ de vecteurs paramètres du problème. Son principe est le suivant: si l'on suppose connaître un certain $\mathbf{B}_{(t)}$, on sait que l'on peut calculer $\mathbf{A}_{(t)}$ par l'équation 9.11, soit ici:

$$\mathbf{A}_{(t)}^T = \mathbf{B}_{(t)}^T M^+ \quad (9.13)$$

On cherche à minimiser le critère $J(\mathbf{A}_{(t)}, \mathbf{B}_{(t)})$. Comment modifier $\mathbf{B}_{(t)}$ pour diminuer cette valeur? Simplement en calculant le gradient de $J(\mathbf{A}_{(t)}, \mathbf{B}_{(t)})$ par rapport à $\mathbf{B}_{(t)}$ et en en déduisant une valeur \mathbf{B}_{t+1} telle que:

$$J(\mathbf{A}_{(t)}, \mathbf{B}_{(t+1)}) \leq J(\mathbf{A}_{(t)}, \mathbf{B}_{(t)}) \quad (9.14)$$

Ce gradient vaut:

$$\nabla_{\mathbf{B}_{(t)}} J(\mathbf{A}_{(t)}, \mathbf{B}_{(t)}) = \nabla_{\mathbf{B}_{(t)}} \|\mathbf{A}_{(t)}^T M - \mathbf{B}_{(t)}^T\| = -2(\mathbf{A}_{(t)}^T M - \mathbf{B}_{(t)}^T) \quad (9.15)$$

Une façon de réduire $J(\mathbf{A}_{(t)}, \mathbf{B}_{(t)})$ est d'utiliser la méthode de la *descente de gradient* (voir l'annexe 18.2), qui consiste ici à retrancher à $\mathbf{B}_{(t)}$ un vecteur colinéaire à $\nabla_{\mathbf{B}_{(t)}} J(\mathbf{A}_{(t)}, \mathbf{B}_{(t)})$, donc à faire:

$$\mathbf{B}_{(t+1)}^T = \mathbf{B}_{(t)}^T + \alpha(\mathbf{A}_{(t)}^T M - \mathbf{B}_{(t)}^T) \quad (9.16)$$

où α est un coefficient positif qui règle la vitesse de convergence de l'algorithme.

Il y a cependant une précaution à prendre: celle d'éviter de rendre négatifs des termes de $\mathbf{B}_{(t+1)}$; pour cela, on remplace simplement dans $(\mathbf{A}_{(t)}^T M - \mathbf{B}_{(t)}^T)$ tous les termes négatifs par 0, ce qui donne un vecteur noté $\lfloor \mathbf{A}_{(t)}^T M - \mathbf{B}_{(t)}^T \rfloor$.

On démontre que cette procédure converge vers une valeur nulle de J quand les deux classes sont séparables, vers une valeur positive sinon; cette dernière valeur peut cependant ne pas être le minimum global de $J(\mathbf{A}, \mathbf{B})$ sur toutes les valeurs possibles de \mathbf{A} et \mathbf{B} . L'algorithme 9.1 décrit cette procédure.

Le critère d'arrêt peut être, par exemple:

$$J(\mathbf{A}_{(t)}, \mathbf{B}_{(t)}) \simeq J(\mathbf{A}_{(t+1)}, \mathbf{B}_{(t+1)})$$

ou bien :

$$t \geq t_{max}$$

La valeur α peut être considérée, du point de vue de l'apprentissage, comme l'importance de la punition ou de la récompense que l'on applique aux paramètres pour les modifier; c'est d'ailleurs une remarque générale pour toutes les méthodes de gradient. Grande, elle permet une bonne exploration de l'espace décrit par J , mais peut faire varier erratiquement les vecteurs $\mathbf{A}_{(t)}$ et

Algorithme 9.1 Algorithme de Ho et Kashyap

```

Prendre  $\mathbf{B}_0$  et  $\alpha$  positif quelconques
 $t \leftarrow 0$ 
tant que critère d'arrêt non satisfait faire
   $\mathbf{A}_{(t)}^T \leftarrow \mathbf{B}_{(t)}^T M^+$ 
   $\mathbf{B}_{(t+1)}^T \leftarrow \mathbf{B}_{(t)}^T + \alpha [\mathbf{A}_{(t)}^T M - \mathbf{B}_{(t)}]^T$ 
   $t \leftarrow t + 1$ 
fin tant que

```

$\mathbf{B}_{(t)}$ et interdire en pratique leur convergence ; petite, elle donne une vitesse de convergence lente et peut mener trop facilement à un optimum local.

Il existe diverses possibilités pour régler automatiquement α qui ont été largement étudiées dans le cadre des méthodes générales d'optimisation par les techniques de descente de gradient. L'une d'entre elles a par exemple pour principe de réduire cette valeur au fur et à mesure de la croissance de t : on peut la voir comme un survol exploratoire à grande vitesse du "paysage" J , progressivement remplacé par une descente de plus en plus lente vers le minimum entrevu comme le plus prometteur¹.

9.2.3 Un autre calcul : l'algorithme du perceptron

Bien que désormais peu utilisé en pratique, cet algorithme est présenté ici à deux titres. D'abord pour son intérêt historique, puisqu'il date pratiquement de l'invention de l'ordinateur et qu'il a été présenté à sa naissance comme une bonne manière de simuler les facultés d'apprentissage². Ensuite et surtout parce qu'il est à la base des méthodes *connexionsnistes* étudiées dans le chapitre 10. Nous avons déjà fait allusion au perceptron dans le chapitre 1.

Cet algorithme travaille directement sur le vecteur \mathbf{a} qui caractérise la surface discriminante cherchée. On n'a donc plus besoin ici de se placer dans l'espace de représentation de dimension $d + 1$ ni d'utiliser le vecteur \mathbf{A} . Cet algorithme travaille de façon itérative : il prend les données d'apprentissage les unes après les autres, chacune étant choisie soit par un passage systématique dans l'ensemble d'apprentissage (version « non-stochastique »), soit par un tirage au hasard dans celui-ci (version « stochastique »). Son nombre d'étapes effectives peut être important : un seul (exactement ou en moyenne) passage des données n'est en effet pas suffisant pour le faire converger.

9.2.3.1 L'algorithme stochastique

Dans sa version stochastique, l'algorithme du perceptron se décrit par l'algorithme 9.2 : à l'étape t de son calcul, le vecteur \mathbf{a} est noté $\mathbf{a}_{(t)}$; le nombre maximal d'étapes est fixé à t_{max} .

On constate qu'il n'y a pas de modification si la donnée lue est classée correctement. En revanche, en cas de mauvais classement, la valeur courante $\mathbf{a}_{(t)}$ de \mathbf{a} est corrigée.

-
1. Ou comme la tactique normale d'un joueur de golf, qui joue des clubs de moins en moins « longs » pour être plus précis au fur et à mesure qu'il se rapproche du trou.
 2. « Le perceptron a appris à lire », tel était le titre d'un article de la revue de vulgarisation scientifique *Atomes* de février 1962.

Algorithme 9.2 Le perceptron, version stochastique.

```

Prendre  $\mathbf{a}_{(0)}$  et  $\alpha$  positif quelconques
 $t \leftarrow 0$ 
tant que  $t \leq t_{max}$  faire
    tirer au hasard une donnée d'apprentissage  $\mathbf{x}$ 
    si  $\mathbf{x}$  est bien classé alors
         $\mathbf{a}_{(t+1)} \leftarrow \mathbf{a}_{(t)}$ 
    sinon
        si  $\mathbf{x} \in \omega_1$  alors
             $\mathbf{a}_{(t+1)} \leftarrow \mathbf{a}_{(t)} + \alpha \mathbf{x}$ 
        sinon
             $\mathbf{a}_{(t+1)} \leftarrow \mathbf{a}_{(t)} - \alpha \mathbf{x}$ 
        fin si
    fin si
     $t \leftarrow t + 1$ 
fin tant que

```

9.2.3.2 Convergence

Le calcul suivant montre empiriquement en quoi cette tactique est fondée : supposons que la donnée d'apprentissage \mathbf{x} appartienne à la classe ω_1 ; on sait que dans ce cas, puisqu'il est mal classé, $\mathbf{A}_{(t)}^T \mathbf{x}$ est négatif au lieu d'être positif.

$$\mathbf{a}_{(t+1)}^T \mathbf{x} = (\mathbf{a}_{(t)} + \alpha \mathbf{x})^T \mathbf{x} = \mathbf{a}_{(t)}^T \mathbf{x} + \alpha \mathbf{x}^T \mathbf{x} = \mathbf{a}_{(t)}^T \mathbf{x} + \alpha \|\mathbf{x}\| \geq \mathbf{a}_{(t)}^T \mathbf{x} \quad (9.17)$$

Par conséquent, puisque la valeur $\mathbf{a}^T \mathbf{x}$ a augmenté, la donnée \mathbf{x} devrait avoir à son prochain passage plus de chances de vérifier l'inégalité $\mathbf{a}^T \mathbf{x} \geq 0$ et d'être donc bien classée dans la classe ω_1 . Le calcul est analogue dans le cas où une donnée devant appartenir à ω_2 est mal classée : la valeur $\mathbf{a}_{(t+1)}^T \mathbf{x}$ devient inférieure à $\mathbf{a}_{(t)}^T \mathbf{x}$, ce qui la rapproche de la valeur négative qu'elle aurait dû prendre.

Cette justification est en fait rigoureusement démontrée par un théorème de convergence qui, comme dans le cas précédent, assure que dans le cas de classes séparables, un hyperplan convenable est trouvé en un nombre fini d'étapes.

9.2.3.3 La version non-stochastique et son interprétation

Reprendons cet algorithme dans sa version non stochastique, dans laquelle les données d'apprentissage sont proposées dans l'ordre. L'itération centrale de l'algorithme (appelée une *époque* d'apprentissage) porte sur le calcul de la modification (notée ci-dessous *modif*) que subit \mathbf{a} par le cumul des contributions de tous les exemples. La base des exemples intervient plusieurs fois. L'algorithme 9.3 décrit le processus.

On peut voir cet algorithme dans cette version comme la minimisation du critère

$$J(\mathbf{A}) = - \sum_{\mathbf{x} \in \mathcal{S} \text{ mal classé par } \mathbf{A}} \mathbf{A}^T \mathbf{x}$$

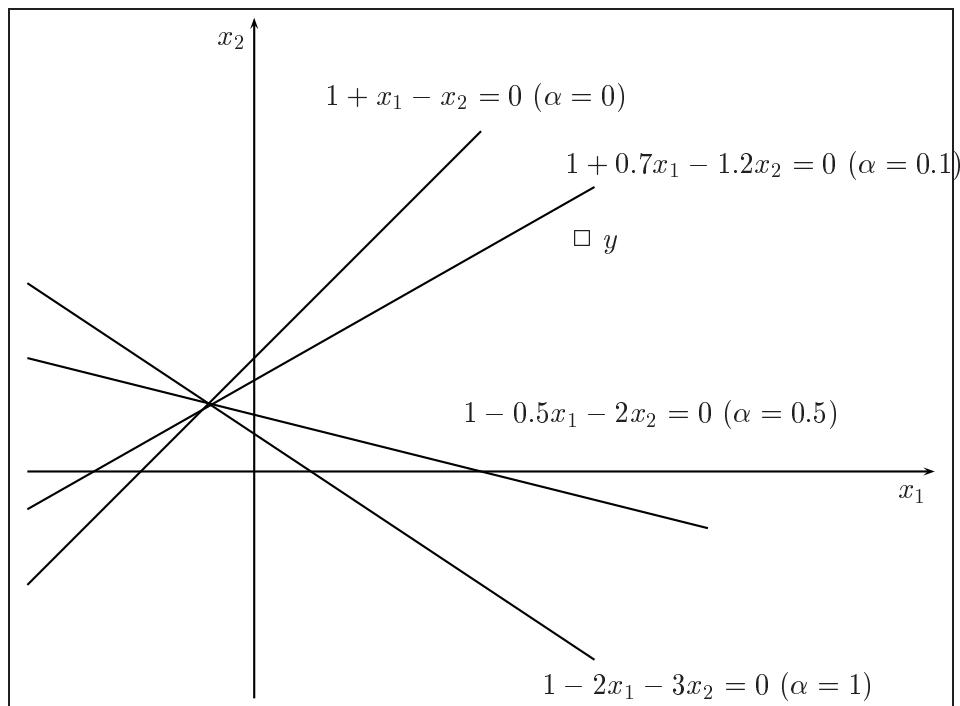


FIG. 9.3 – Fonctionnement de l'algorithme du perceptron.

En effet, en supposant a_0 fixé à la valeur 1, on a :

$$\nabla_{\mathbf{a}} J(\mathbf{a}) = - \sum_{\mathbf{x} \in \mathcal{S} \text{ mal classé par } \mathbf{a}} \mathbf{x} = \frac{\text{modif}}{\alpha}$$

Par conséquent, la modification apportée au vecteur \mathbf{A} par une époque d'apprentissage est proportionnelle au gradient du critère $J(\mathbf{a})$. Cet algorithme minimise donc $J(\mathbf{a})$, qui est un critère un peu différent de celui de l'algorithme de Ho et Kashyap. L'expérience prouve en général que ce dernier est plus efficace du point de vue de la généralisation.

9.2.3.4 Fonctionnement sur un exemple

Prenons l'espace de représentation égal à \mathbb{R}^2 (figure 9.3) et supposons qu'à l'itération t , l'équation de la surface séparatrice courante soit :

$$x_1 - x_2 + 1 = 0$$

Autrement dit, on a fixé $a_0 = 1$ et le vecteur courant $\mathbf{a}_{(t)}$ vaut : $\mathbf{a}_{(t)} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$.

Maintenant, supposons que l'exemple suivant soit : $(\mathbf{x}, c) = \begin{pmatrix} 3 \\ 2 \\ \omega_2 \end{pmatrix}$

La valeur de l'équation de la séparatrice $x_1 - x_2 + 1 = 0$ sur les coordonnées de l'exemple est donc :

$$3 - 2 + 1 = 2$$

Or, puisque l'exemple (\mathbf{x}, c) appartient à la classe ω_2 , cette valeur devrait être négative. Il faut donc modifier $\mathbf{a}_{(t)}$.

Algorithme 9.3 Le perceptron, version non stochastique.

Prendre $A_{(0)}$ quelconque et α positif quelconque

$t \leftarrow 0$

tant que $t \leq t_{max}$ **faire**

$modif \leftarrow 0$

pour chaque donnée d'apprentissage x **faire**

si x est mal classé **alors**

si $x \in \omega_1$ **alors**

$modif \leftarrow modif + \alpha x$

sinon

$modif \leftarrow modif - \alpha x$

fin si

fin si

fin pour

$A_{(t+1)} \leftarrow A_{(t)} + modif$

$t \leftarrow t + 1$

fin tant que

Le calcul à faire, puisque $(x, c) \in \omega_2$, est le suivant :

$$\mathbf{a}_{(t+1)} = \mathbf{a}_{(t)} - \alpha \cdot x$$

Pour $\alpha = 0.1$, on obtient :

$$\mathbf{a}_{(t+1)} = \mathbf{a}_{(t)} - 0.1 \begin{pmatrix} 3 \\ 2 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \end{pmatrix} - \begin{pmatrix} 0.3 \\ 0.2 \end{pmatrix} = \begin{pmatrix} 0.7 \\ -1.2 \end{pmatrix}$$

La surface séparatrice courante a donc pour nouvelle équation :

$$0.7x_1 - 1.2x_2 + 1 = 0$$

On constate graphiquement qu'elle s'est rapprochée du nouvel exemple, sans toutefois bien le classer.

Pour un pas plus grand, par exemple : $\alpha = 1$, on obtiendrait la séparatrice :

$$-2x_1 - 3x_2 + 1 = 0$$

qui classe correctement le nouveau point.

On verra au chapitre 10 une extension du perceptron au cas de plus de deux classes ; on verra aussi comment on peut le généraliser en réseau connexionniste.

9.2.4 L'hyperplan discriminant de Fisher

La méthode de Fisher diffère des précédentes en ce qu'elle ramène d'abord le problème à une dimension avant de construire un hyperplan. Pour ce faire, elle va chercher la droite F de vecteur directeur f passant par l'origine telle que la projection des points d'apprentissage sur F sépare au mieux les deux classes ω_1 et ω_2 . Ensuite, elle cherche sur F le point par lequel passe l'hyperplan discriminant, orthogonal à F .

Notons $\boldsymbol{\mu}_1$ et $\boldsymbol{\mu}_2$ la moyenne des points d'apprentissage de ω_1 de ω_2 , m_1 et m_2 le nombre de points d'apprentissage dans ω_1 et ω_2 et définissons :

$$\overline{\mu_1} = \frac{1}{m_1} \sum_{\mathbf{x} \in \omega_1} \mathbf{f}^T \mathbf{x} \quad (9.18)$$

comme le scalaire donnant la moyenne des valeurs des projections des points de ω_1 sur F , et notons de même $\overline{\mu_2}$ pour la seconde classe. Définissons aussi :

$$\overline{s_1} = \sum_{\mathbf{x} \in \omega_1} (\mathbf{f}^T \mathbf{x} - \overline{\mu_1})^2 \quad (9.19)$$

et $\overline{s_2}$ la *dispersion* de ces projections. Le critère de Fisher consiste à chercher la droite F telle que la valeur

$$J(F) = \frac{(\overline{\mu_1} - \overline{\mu_2})^2}{\overline{s_1}^2 + \overline{s_2}^2} \quad (9.20)$$

soit maximale. Cette valeur est en effet une mesure de la séparation des projections des deux classes sur la droite F . On peut d'ailleurs l'exprimer autrement.

Définissons les valeurs :

$$S_1 = \sum_{\mathbf{x} \in \omega_1} (\mathbf{x} - \boldsymbol{\mu}_1)(\mathbf{x} - \boldsymbol{\mu}_1)^T \quad (9.21)$$

et de même S_2 ; la valeur

$$S_I = S_1 + S_2 \quad (9.22)$$

est appelée la *variance intraclasse totale* des données d'apprentissage. Définissons aussi la *variance interclasse* par :

$$S_J = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T \quad (9.23)$$

Il est alors possible de prouver que :

$$J(F) = \frac{\mathbf{f}^T S_J \mathbf{f}}{\mathbf{f}^T S_I \mathbf{f}} \quad (9.24)$$

Finalement, on démontre que le critère $J(F)$ est maximal pour :

$$\widehat{\mathbf{f}} = S_I^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \quad (9.25)$$

L'hyperplan séparateur de Fisher étant orthogonal à la droite F , il a donc pour équation :

$$\widehat{\mathbf{f}} \mathbf{x} - f_0 = 0$$

La dernière inconnue est la valeur scalaire f_0 . Puisque le problème a été ramené à une dimension, il suffit pour la déterminer de supposer par exemple que les projections de chaque classe sur F sont gaussiennes et de procéder comme au chapitre 14.

9.2.5 Surfaces séparatrices non linéaires

On pourrait assez facilement étendre les calculs et les algorithmes précédents à la recherche de surfaces discriminantes non linéaires ; cependant, cette approche est rarement poursuivie, pour deux raisons. D'abord, compliquer les surfaces reviendrait à augmenter le nombre de paramètres à apprendre, ce qui mène à une perte de précision dans leur estimation, pour un nombre d'exemples donné. Et surtout, en appliquant ainsi le principe *ERM* sans contrôler la complexité des hypothèses, on arriverait en augmentant trop le degré de la surface à un apprentissage par cœur. Cette situation est telle que la règle apprise fournit une probabilité d'erreur apparente nulle, mais classe mal des exemples de test pourtant issus des mêmes processus aléatoires que les données d'apprentissage (voir le chapitre 3). Pour ces raisons, la recherche de surfaces discriminantes se limite en pratique à celle d'hyperplans. Nous verrons au chapitre 14 que les surfaces séparatrices implicitement calculées par la méthode du plus proche voisin (voir le chapitre 14) et par les réseaux connexionnistes à couches cachées (chapitre 10) sont des *hyperplans par morceaux* (en dimension 2, des “lignes brisées”). On n'a donc en général pas besoin de pousser la recherche directe de séparatrices au-delà de celle des surfaces linéaires.

9.2.6 Et pour plus de deux classes ?

Dans le cas où l'on a un ensemble d'apprentissage représentatif de plus de deux classes, il existe plusieurs façons de généraliser la discrimination linéaire. La première est d'apprendre pour chaque classe un hyperplan qui la discrimine de toutes les autres, ce qui revient à considérer la réunion de leurs exemples comme négatifs et ceux de la classe en apprentissage comme positifs. Mais cette façon de faire conduit, comme le montre la figure 9.4, à de larges ambiguïtés de classification.

Une autre solution est de chercher une surface discriminante entre chaque couple de classes (figure 9.5) ; elle présente l'avantage de dessiner moins de zones ambiguës, mais elle a l'inconvénient de demander le calcul de $C(C - 1)/2$ jeux de paramètres au lieu de C .

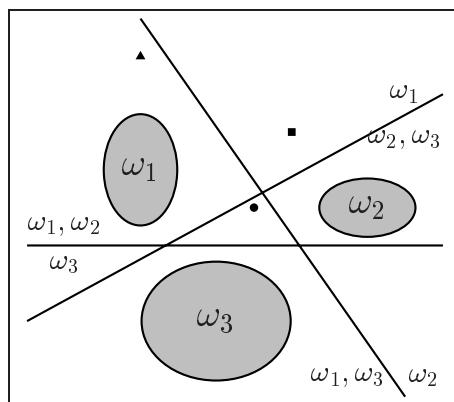


FIG. 9.4 – Séparation linéaire à plus de deux classes. On sépare chaque classe de toutes les autres : il y a C hyperplans. Le point en triangle est attribué à la classe ω_1 , le point en carré est ambigu entre ω_1 et ω_2 , le point central est ambigu entre les trois classes. Sur les sept zones, quatre sont ambiguës.

Dans le cas de plus de trois classes, la géométrie de la séparation par hyperplans devient nettement plus complexe : l'espace \mathbb{R}^d est en effet partagé dans le cas général en $\frac{n^2+n+2}{2}$ zones convexes par n hyperplans.

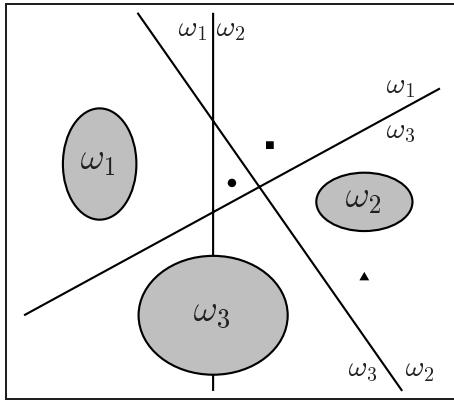


FIG. 9.5 – Séparation linéaire à plus de deux classes. On sépare chaque classe de chaque autre : il y a $\frac{C(C-1)}{2}$ hyperplans. Le point en triangle et le point en carré sont attribués à la classe ω_2 , le point central est ambigu entre les trois classes. Sur les sept zones, une seule est ambiguë.

Les problèmes d'ambiguïté peuvent être résolus par un système ultérieur de décision sur l'ensemble des classes, prenant en compte des paramètres supplémentaires, par exemple la distance aux hyperplans appris (qui se déduit directement des formules présentées ci-dessus) et/ou les probabilités *a priori* des classes. On peut aussi regrouper provisoirement les classes par des méthodes de classification hiérarchique, et réaliser un processus de décision arborescent sur cette base. Ceci n'est pas sans rapport avec les méthodes des *arbres de décision* présentées au chapitre 11. Notons aussi que l'algorithme du perceptron, donné ici dans le cas de deux classes, possède une généralisation naturelle à plus de deux classes qui sera présentée au chapitre 10.

9.3 Les séparateurs à vastes marges (SVM)

Ce chapitre est dédié à la recherche du meilleur hyperplan permettant de discriminer deux classes. Dans les sections précédentes, le critère de performance était lié au nombre d'exemples incorrectement classés, et l'apprenant cherchait une fonction de décision du type

$$g(\mathbf{x}) = \text{sign}\left(\sum_{i=0}^d w_i x_i\right)$$

c'est-à-dire une fonction indicatrice. Dans ce cas-là, la seule information que l'on obtienne sur l'entrée est qu'elle est d'un côté ou de l'autre de la frontière de décision. Si l'on considère maintenant la distance (ou une notion apparentée) des exemples à l'hyperplan séparateur, et des fonctions de décision du type $h(\mathbf{x}) = \sum_{i=0}^d w_i x_i$, est-ce que l'apprentissage devient plus performant puisqu'il prend en compte une information supplémentaire sur les données ? La suite de ce chapitre est dédiée à cette question. De fait, une grande partie des méthodes récentes et des travaux théoriques en cours concernent l'apprentissage de *séparateurs à vastes marges* (SVM), terme que nous avons préféré à celui de machines à vecteurs de support qui est la traduction littérale des *Support Vector Machines* développées originellement par Vapnik dans les années 1980 et 1990.

Dans cette section, nous présenterons essentiellement la technique des séparateurs à vastes marges dans le cadre de la classification supervisée binaire, donc de la séparation de deux classes.

Il existe des généralisations multiclass de ces techniques, de même que tout un ensemble de travaux sur l'application de ces méthodes à la tâche de régression. Dans l'espace limité de cet ouvrage, nous en parlerons peu. Pour la même raison, nous nous tiendrons aux idées essentielles et nous reportons les lecteurs intéressés à la littérature foisonnante sur le sujet, en particulier [CST00, Her02, SBE99, SS02, SBSE00, Vap95] et [AB96] pour une analyse théorique poussée.

Dans un premier temps, nous présentons la « machinerie » des séparateurs à vastes marges, sans nous préoccuper de leur justification qui sera l'objet de la deuxième partie. Nous terminerons en évoquant les extensions de ces méthodes à la classification multiclass et à la régression, et en soulignant les liens avec d'autres méthodes également très en vogue comme les méthodes à base de fonctions noyau ou le *boosting* qui est présenté dans le chapitre 11.

9.3.1 La recherche des séparateurs linéaires à vastes marges

9.3.1.1 Séparateurs linéaires et marge

Supposons donné l'échantillon d'apprentissage: $\mathcal{S} = \{(\mathbf{x}_1, u_1), \dots, (\mathbf{x}_m, u_m)\}$, avec $\mathbf{x}_i \in \mathbb{R}^d$, $u_i \in \{-1, 1\}$. On suppose dans un premier temps qu'il existe une séparatrice linéaire permettant de distinguer les exemples positifs (étiquetés *VRAI* ou +1) des exemples négatifs (étiquetés *FAUX* ou -1). On sait que la recherche d'une telle séparatrice dans \mathcal{X} revient à chercher une fonction hypothèse $h(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + w_0$ telle que:

$$\mathbf{w}^\top \mathbf{x}_i + w_0 \begin{cases} > 0 \\ < 0 \end{cases} \implies u_i = \begin{cases} +1 \\ -1 \end{cases}$$

Cette séparatrice est valide sur l'échantillon d'apprentissage si: $\forall 1 \leq i \leq m, u_i h(\mathbf{x}_i) > 0$. Soit encore si:

$$\forall 1 \leq i \leq m, u_i (\mathbf{w}^\top \mathbf{x}_i + w_0) > 0.$$

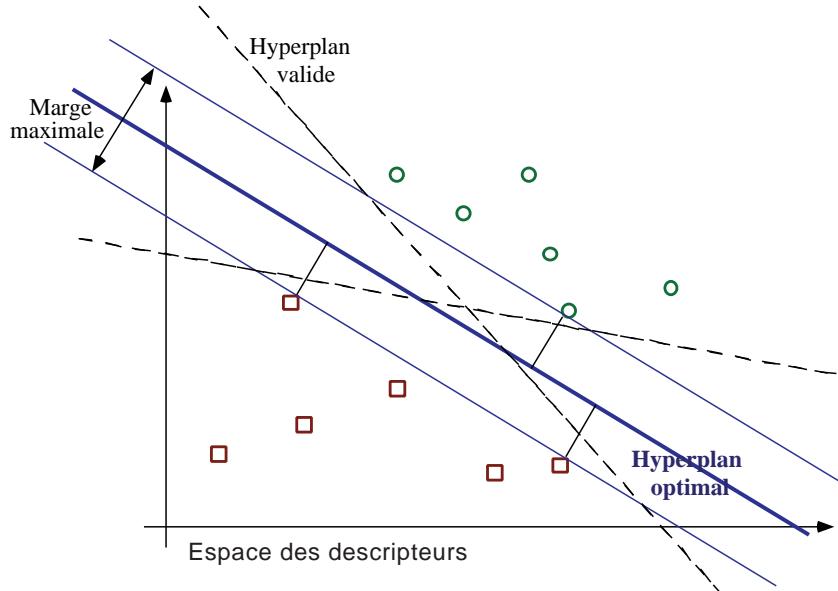


FIG. 9.6 – Séparation de deux ensembles de points par des séparatrices linéaires. L'hyperplan se trouvant « au milieu » des deux nuages de points est appelé hyperplan optimal.

La fonction $h(\mathbf{x})$ correspond à l'équation d'un hyperplan dans \mathcal{X} de vecteur normal \mathbf{w} . La distance d'un point \mathbf{x} à l'hyperplan d'équation $h(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + w_0$ est égale à: $h(\mathbf{x})/\|\mathbf{w}\|$, où

$\| \mathbf{w} \|$ est la norme euclidienne du vecteur \mathbf{w} . Lorsqu'il existe une séparatrice linéaire entre les points d'apprentissage, il en existe en général une infinité. On peut alors chercher parmi ces séparatrices celle qui est « au milieu » des deux nuages de points exemples et contre-exemples. Cet hyperplan optimal est défini par :

$$\underset{\mathbf{w}, w_0}{\operatorname{Argmax}} \min\{\|\mathbf{x} - \mathbf{x}_i\| : \mathbf{x} \in \mathbb{R}^d, (\mathbf{w}^\top \mathbf{x} + w_0) = 0, i = 1, \dots, m\}$$

c'est-à-dire l'hyperplan qui maximise la distance minimale aux exemples d'apprentissage (voir figure 9.7). Dans ce cas, la marge vaut : $2/\|\mathbf{w}\|$.

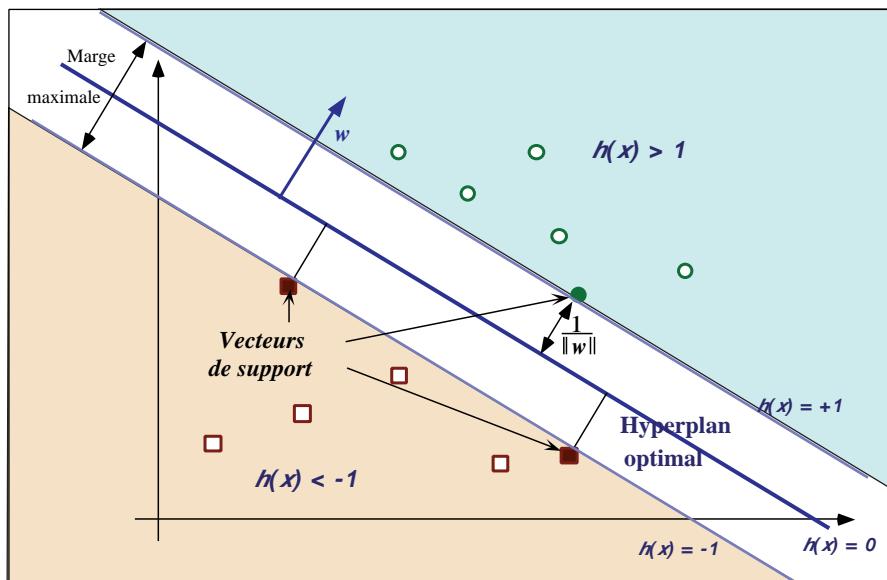


FIG. 9.7 – L'hyperplan optimal est perpendiculaire au segment de droite le plus court joignant un exemple d'apprentissage à l'hyperplan. Ce segment a pour longueur $\frac{1}{\|\mathbf{w}\|}$ lorsqu'on normalise convenablement les paramètres \mathbf{w} et w_0 .

9.3.1.2 L'expression primale du problème des SVM

La recherche de l'hyperplan optimal revient donc à minimiser $\|\mathbf{w}\|$, soit à résoudre le problème d'optimisation suivant qui porte sur les paramètres \mathbf{w} et w_0 :

$$\begin{cases} \text{Minimiser} & \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{sous les contraintes} & u_i(\mathbf{w}^\top \mathbf{x}_i + w_0) \geq 1 \quad i = 1, \dots, m \end{cases} \quad (9.26)$$

Cette écriture du problème, appelée *formulation primale*, implique le réglage de $d + 1$ paramètres, d étant la dimension de l'espace des entrées \mathcal{X} . Cela est possible avec des méthodes de programmation quadratique³ pour des valeurs de d assez petites, mais devient inenvisageable pour des valeurs de d dépassant quelques centaines. Heureusement, il existe une transformation de ce problème dans une formulation duale que l'on peut résoudre en pratique.

3. Les problèmes d'optimisation pour lesquels la fonction et les contraintes sont linéaires ressortent des techniques de programmation linéaire, tandis que les problèmes d'optimisation dans lesquels la fonction est quadratique et les contraintes linéaires ressort des techniques de la programmation quadratique : voir le chapitre 3.

9.3.1.3 L'expression duale du problème de recherche des SVM

D'après la théorie de l'optimisation, un problème d'optimisation possède une forme duale dans le cas où la fonction objectif et les contraintes sont strictement convexes. Dans ce cas, la résolution de l'expression duale du problème est équivalente à la solution du problème original. Ces critères de convexité sont réalisés dans le problème (9.26). Pour résoudre ces types de problèmes, on utilise une fonction que l'on appelle *lagrangien* qui incorpore des informations sur la fonction objectif et sur les contraintes et dont le caractère stationnaire peut être utilisé pour détecter des solutions. Plus précisément, le lagrangien est défini comme étant la somme de la fonction objectif et d'une combinaison linéaire des contraintes dont les coefficients $\alpha_i \geq 0$ sont appelés *multiplicateurs de Lagrange* ou encore *variables duales*.

$$L(\mathbf{w}, w_0, \boldsymbol{\alpha}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^m \alpha_i (u_i \cdot (\mathbf{x}_i \cdot \mathbf{w}) + w_0) - 1 \quad (9.27)$$

Une théorème de Kuhn-Tucker, qui couronne des travaux commencés par Fermat, puis poursuivis par Lagrange, démontre que le problème primal et sa formulation duale ont la même solution qui correspond à un *point-selle* du lagrangien (il faut le minimiser par rapport aux variables primaires \mathbf{w} et w_0 et le maximiser par rapport aux variables duales α_i).

Au point-selle, la dérivée du Lagrangien par rapport aux variables primaires doit s'annuler. Ceci s'écrit :

$$\frac{\partial}{\partial \mathbf{w}} L(\mathbf{w}, w_0, \boldsymbol{\alpha}) = 0, \quad \frac{\partial}{\partial w_0} L(\mathbf{w}, w_0, \boldsymbol{\alpha}) = 0 \quad (9.28)$$

et conduit à :

$$\sum_{i=1}^m \alpha_i u_i = 0 \quad (9.29)$$

et à :

$$\mathbf{w} = \sum_{i=1}^m \alpha_i u_i \mathbf{x}_i \quad (9.30)$$

Par ailleurs, il est montré (*conditions de Karush-Kuhn-Tucker*) que seuls les points qui sont sur les hyperplans frontière $(\mathbf{x}_i \cdot \mathbf{w}) + w_0 = \pm 1$ jouent une rôle. Ces points pour lesquels les multiplicateurs de Lagrange sont non nuls sont appelés *vecteurs de support* par Vapnik. Nous utiliserons aussi le terme plus imagé d'*exemples critiques* puisque ce sont eux qui déterminent l'hyperplan optimal.

Il est ainsi remarquable que le vecteur solution \mathbf{w}^* ait une expression en termes d'un sous-ensemble des exemples d'apprentissage: les exemples critiques. C'est en même temps intuitivement satisfaisant puisque l'on « voit » bien que l'hyperplan solution est entièrement déterminé par ces exemples (voir figure 9.7). C'est intuitif, oui, mais c'est remarquable.

En substituant (9.29) et (9.30) dans (9.28), on élimine les variables primaires et l'on obtient la *forme duale* du problème d'optimisation :

trouver les multiplicateurs de Lagrange tels que :

$$\begin{cases} \text{Max}_{\boldsymbol{\alpha}} \left\{ \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m \alpha_i \alpha_j u_i u_j (\mathbf{x}_i \cdot \mathbf{x}_j) \right\} \\ \alpha_i \geq 0, \quad i = 1, \dots, m \\ \sum_{i=1}^m \alpha_i u_i = 0 \end{cases} \quad (9.31)$$

L'hyperplan solution correspondant peut alors être écrit :

$$h(\mathbf{x}) = (\mathbf{w}^* \cdot \mathbf{x}) + w_0^* = \sum_{i=1}^m \alpha_i^* u_i \cdot (\mathbf{x} \cdot \mathbf{x}_i) + w_0^* \quad (9.32)$$

où les α_i^* sont solution de (9.31) et w_0 est obtenue en utilisant n'importe quel exemple critique (\mathbf{x}_c, u_c) dans l'équation :

$$\alpha_i [u_i \cdot ((\mathbf{x}_i \cdot \mathbf{w}^*) + w_0) - 1] = 0 \quad (9.33)$$

À ce point, remarquons deux choses. D'abord, que l'hyperplan solution ne requiert que le calcul des produits scalaires $\langle \mathbf{x} \cdot \mathbf{x}_i \rangle$ entre des vecteurs de l'espace d'entrée \mathcal{X} . Cette observation aura de profondes répercussions. Ensuite, la solution ne dépend plus de la dimension d de l'espace d'entrée, mais de la taille m de l'échantillon de données et même du nombre m_c d'exemples critiques qui est généralement bien inférieur à m . Les méthodes d'optimisation quadratique standards suffisent donc pour de nombreux problèmes pratiques.

9.3.1.4 La cas d'un échantillon non linéairement séparable

Supposons maintenant que les exemples ne puissent pas être linéairement séparés. Peut-on aménager la méthode des séparateurs à vastes marges (SVM)? Une technique dite des *variables ressort (slack variables)* permet en effet de chercher un hyperplan séparateur faisant le moins d'erreurs possible. Pour cela, on modifie les contraintes en les relâchant grâce à des variables ressort $\xi_i \geq 0$:

$$u_i \cdot ((\mathbf{w} \cdot \mathbf{x}_i) + w_0) \geq 1 - \xi_i \quad (9.34)$$

On doit dans ce cas minimiser :

$$\frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^m \xi_i \quad (9.35)$$

pour une constante $C > 0$.

On obtient alors, comme dans le cas séparable, une formulation duale, excepté les contraintes qui sont légèrement différentes.

$$\begin{cases} \text{Max}_{\boldsymbol{\alpha}} \left\{ \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j u_i u_j (\mathbf{x}_i \cdot \mathbf{x}_j) \right\} \\ \forall i, \quad 0 \leq \alpha_i \leq C \\ \sum_{i=1}^m \alpha_i u_i = 0 \end{cases} \quad (9.36)$$

Le coefficient C règle le compromis entre la marge possible entre les exemples et le nombre d'erreurs admissibles. Il doit être choisi par l'utilisateur⁴.

4. La description faite ici concerne la minimisation du nombre d'exemples qui ne sont pas du bon côté des marges. Il existe une autre formulation qui prend en compte non le nombre d'erreurs, mais la distance euclidienne à la marge des exemples mal positionnés. Nous renvoyons le lecteur à [CST00] pp.103-110 par exemple pour plus de détails.

9.3.1.5 Le passage par un espace de redescription : les fonctions noyau

Une motivation majeure des séparateurs à vastes marges (SVM) est que le problème d'optimisation correspondant n'a pas d'optima locaux mais seulement un optimum global, facile à trouver qui plus est. Cependant, l'expressivité limitée des séparateurs linéaires semble devoir les confiner à des problèmes très particuliers, comme l'avaient fort bien montré en leur temps Minsky et Papert dans leur célèbrissime ouvrage *Perceptrons* ([MP69]). D'où le recours aux perceptrons multicouches mis au point dans les années quatre-vingt (voir chapitre 10), mais qui admettent généralement plusieurs optima locaux, difficiles à déterminer de surcroît. Ne peut-on donc avoir le meilleur des deux mondes ?

Un pas dans cette direction est fait quand on réalise qu'un problème de discrimination non linéaire peut se transformer en problème de séparation linéaire si l'on dispose des bons descripteurs pour décrire les données. Encore faut-il connaître ces descripteurs, ce qui revient souvent en pratique à connaître la solution. Pourtant, une approche combinatoire permet d'imaginer une telle transformation sans connaissance *a priori* sur le problème. En effet, plus la dimension de l'espace de description est grande, plus la probabilité de pouvoir trouver un hyperplan séparateur entre les exemples et les contre-exemples est élevée. En transformant l'espace d'entrée en un espace de redescription de très grande dimension, éventuellement infinie, il devient donc possible d'envisager d'utiliser la méthode des SVM.

Notons Φ une transformation non linéaire de l'espace d'entrée \mathcal{X} en un *espace de redescription* $\Phi(\mathcal{X})$:

$$\mathbf{x} = (x_1, \dots, x_d)^\top \rightarrow \Phi(\mathbf{x}) = (\phi_1(\mathbf{x}), \dots, \phi_d(\mathbf{x}), \dots) \quad (9.37)$$

Le problème d'optimisation se transcrit dans ce cas par :

$$\begin{cases} \text{Max}_{\alpha} \left\{ \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m \alpha_i \alpha_j u_i u_j \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle \right\} \\ \alpha_i \geq 0, \quad i = 1, \dots, m \\ \sum_{i=1}^m \alpha_i u_i = 0 \end{cases} \quad (9.38)$$

où $\langle \cdot, \cdot \rangle$ dénote le produit scalaire⁵.

L'équation de l'hyperplan séparateur dans le nouvel espace devient :

$$h(\mathbf{x}) = \sum_{i=1}^m \alpha_i^* u_i \langle \Phi(\mathbf{x}) . \Phi(\mathbf{x}_i) \rangle + w_0^* \quad (9.39)$$

où les coefficients α_i^* et w_0^* sont obtenus comme précédemment par résolution de (9.38).

Tout cela est très bien, sauf que l'objection immédiate du praticien concerne le produit scalaire $\langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle$ qui devient rapidement impossible à calculer quand la dimension de $\Phi(\mathcal{X})$ augmente (sans parler du cas de la dimension infinie), ceci d'autant plus que l'on utilisera des transformations non linéaires des descripteurs d'entrée. Pour donner une idée, supposons que l'on cherche à classer des images de 16×16 pixels (par exemple pour faire de la reconnaissance d'écriture manuscrite), et que l'on suppose nécessaire pour cela de tenir compte des corrélations entre 5 pixels quelconques au plus dans les images. L'espace de redescription, qui contient toutes les combinaisons de 5 pixels quelconques parmi 256, est alors de dimension de l'ordre de 10^{10} . Calculer des produits scalaires dans un tel espace est impraticable.

Il se trouve heureusement que l'on peut dans certains cas s'arranger pour littéralement court-circuiter le passage par les calculs dans l'espace de redescription. En effet, il existe des fonctions

5. Parfois aussi appelé produit *interne*, de l'anglais *inner product*.

bilinéaires symétriques positives $K(x, y)$, appelées *fonctions noyau*, faciles à calculer et dont on peut montrer qu'elles correspondent à un produit scalaire $\langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle$ dans un espace de grande dimension. Lorsqu'une telle correspondance est exploitable, le problème d'optimisation (9.38) est équivalent au problème suivant :

$$\begin{cases} \text{Max}_{\alpha} \left\{ \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m \alpha_i \alpha_j u_i u_j K(\mathbf{x}_i, \mathbf{x}_j) \right\} \\ \alpha_i \geq 0, \quad i = 1, \dots, m \\ \sum_{i=1}^m \alpha_i u_i = 0 \end{cases} \quad (9.40)$$

dont la solution est l'hyperplan séparateur d'équation :

$$h(\mathbf{x}) = \sum_{i=1}^m \alpha_i^* u_i K(\mathbf{x}, \mathbf{x}_i) + w_0^* \quad (9.41)$$

où les coefficients α_i^* et w_0^* sont obtenus comme précédemment par résolution de (9.40).

Par exemple, il peut être montré que la *fonction noyau polynomiale* :

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^n \quad (9.42)$$

réalise implicitement un produit scalaire dans l'espace des descripteurs correspondant à tous les produits d'exactement n dimensions. Ainsi pour $n = 2$ et $\mathbf{x}, \mathbf{y} \in \mathbb{R}^2$, nous avons :

$$(\mathbf{x} \cdot \mathbf{y})^2 = (x_1^2, x_2^2, \sqrt{2}x_1x_2)(y_1^2, y_2^2, \sqrt{2}y_1y_2)^\top = \langle \Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}) \rangle$$

qui correspond au changement de description par la fonction : $\Phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$.

Autre exemple : la fonction noyau $K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + c)^n$ avec $c > 0$ correspond à un produit scalaire dans l'espace des descripteurs correspondant à tous les produits d'ordre $\leq n$.

Les fonctions noyau décrites impliquent des calculs dans l'espace \mathcal{X} , donc dans un espace de dimension d . Ils sont par conséquent faciles à calculer. Mais comment déterminer quelle fonction noyau aisément calculable est associée à un espace de redescription $\Phi(\mathcal{X})$ dont on pense qu'il peut être intéressant pour trouver un séparateur linéaire des données ?

En fait, la démarche est inverse : on cherche des fonctions noyau dont on peut avoir la garantie *a priori* qu'elles correspondent à un produit scalaire dans un certain espace qui agira comme un espace de redescription, mais qui restera virtuel, jamais explicité. Cela signifie que l'utilisateur devra opérer par essais et erreurs : essayer des fonctions noyau associées à des produits scalaires dans des espaces de redescription et voir si elles permettent l'obtention de bonnes fonctions de décision. Si cet aspect tâtonnant de la méthode peut sembler peu satisfaisant, en revanche, le choix de la fonction noyau devient le seul paramètre à régler, contrairement à d'autres techniques d'apprentissage.

Sous quelles conditions une fonction noyau est-elle équivalente à un produit scalaire dans un espace⁶ ?

6. Les concepts mis en jeu pour répondre à cette question impliquent une assez grande familiarité avec les mathématiques et particulièrement avec l'*analyse fonctionnelle* développée par David Hilbert (1862-1943). Il ne s'agit pas de s'étendre ici sur cette théorie, mais pas non plus de l'escamoter car il est probable que sa compréhension profonde peut conduire à de futurs développements en apprentissage. L'idée essentielle est d'étudier les espaces de fonctions lorsqu'on peut les doter d'un produit scalaire et donc d'une distance, ce qui ouvre la porte à tous les problèmes d'approximation, et d'équations, sur les fonctions prises comme objets (vecteurs d'un espace de Hilbert). Parmi ces équations, il en est qui jouent un grand rôle en physique, ce sont les *équations intégrales* du type :

$$f(x) = \lambda \int_a^b K(x, y) f(y) dy$$

dans lesquelles l'inconnue est la fonction $f(x)$, et $K(x, y)$ est le noyau de l'équation intégrale.

Soit donc la redescription des formes \mathbf{x} de \mathcal{X} par la fonction Φ dans un espace de Hilbert de coordonnées $\phi_1(\mathbf{x}), \dots, \phi_d(\mathbf{x})$. D'après la théorie de Hilbert-Schmidt, le produit scalaire dans l'espace de Hilbert possède une représentation équivalente :

$$\langle \Phi(\mathbf{x}) . \Phi(\mathbf{x}') \rangle = \sum_{i=1}^{\infty} \lambda_i \phi_i(\mathbf{x}) \phi_i(\mathbf{x}') = K(\mathbf{x}, \mathbf{x}') \quad (9.43)$$

avec $\lambda_i \geq 0, \forall i$, si la fonction $K(\mathbf{x}, \mathbf{x}')$ est une fonction symétrique satisfaisant aux conditions de Mercer énoncées dans le théorème du même nom.

Théorème 9.1 (Théorème de Mercer)

Si $K(.,.)$ est une fonction noyau continue symétrique d'un opérateur intégral

$$g(\mathbf{y}) = A f(\mathbf{y}) = \int_a^b K(x, y) f(y) dy + h(x)$$

vérifiant :

$$\int_{\mathcal{X} \times \mathcal{X}} K(\mathbf{x}, \mathbf{x}') f(\mathbf{x}) f(\mathbf{x}') d\mathbf{x} d\mathbf{x}' \geq 0$$

pour toute fonction $f \in L_2(\mathcal{X})$ (de carré sommable) (\mathcal{X} étant un sous-espace compact de \mathbb{R}^d), alors la fonction $K(.,.)$ peut être développée en une série uniformément convergente en fonction des valeurs propres positives λ_i et des fonctions propres ψ_i :

$$K(\mathbf{x}, \mathbf{x}') = \sum_{j=1}^N \lambda_j \psi_j(\mathbf{x}) \psi_j(\mathbf{x}') \quad (9.44)$$

où N est le nombre de valeurs propres positives.

Le théorème de Mercer donne donc les conditions pour qu'une fonction $K(.,.)$ soit équivalente à un produit scalaire (9.44).

Une condition équivalente à celle de Mercer est que les matrices $K_{ij} := K(\mathbf{x}_i, \mathbf{x}_j)$ soient positives semi-définies⁷ pour tout échantillon d'exemples $\{\mathbf{x}_1, \dots, \mathbf{x}_m\}$ (ces matrices sont appelées matrices de Gram).

On peut alors décrire la fonction Φ de redescription des entrées comme :

$$\Phi(\mathbf{x}) = (\sqrt{\lambda_1} \psi_1(\mathbf{x}), \sqrt{\lambda_2} \psi_2(\mathbf{x}), \dots) \quad (9.45)$$

Si les conditions théoriques exposées ci-dessus sont intéressantes, elles ne sont pas faciles à vérifier en pratique. La plupart des praticiens utilisent l'une des fonctions noyau connues (voir le tableau ci-dessous) ou bien font usage de la propriété (9.2) pour construire de nouvelles fonctions noyau adaptées à leur problème.

7. Dont les valeurs propres sont toutes positives ou nulles.

La solution s'exprime sous la forme :		
- Polynomiale	$h(\mathbf{x}) = \sum_{i=1}^{n_{\text{support}}} \alpha_i^* u_i K(\mathbf{x}, \mathbf{x}_i) + w_0^*$	La puissance p est déterminée <i>a priori</i> par l'utilisateur.
- Fonctions à Base Radiale (RBF)	$K(\mathbf{x}, \mathbf{x}') = e^{-\frac{\ \mathbf{x}-\mathbf{x}'\ ^2}{2\sigma^2}}$	L'écart-type σ^2 , commun à tous les noyaux, est spécifié <i>a priori</i> par l'utilisateur.
- Fonctions sigmoïdes	$K(\mathbf{x}, \mathbf{x}') = \tanh(a(\mathbf{x} \cdot \mathbf{x}' - b))$	Le théorème de Mercer n'est vérifié que pour certaines valeurs de a et b .

Théorème 9.2 (Construction de fonctions noyau à partir de fonctions noyau)

Soient K_1 et K_2 deux fonctions noyau définies sur $\mathcal{X} \times \mathcal{X}$, $\mathcal{X} \in \mathbb{R}^d$, $a \in \mathbb{R}^+$, $f(\cdot)$ une fonction réelle sur \mathcal{X} , $\Phi : \mathcal{X} \rightarrow \mathbb{R}^l$, K_3 une fonction noyau définie sur $\mathbb{R}^l \times \mathbb{R}^l$, \mathbf{K} une matrice $d \times d$ symétrique positive et semi-définie et \mathbf{B} une matrice diagonalisable. Alors les fonctions suivantes sont des fonctions noyau :

1. $K(\mathbf{x}, \mathbf{z}) = K_1(\mathbf{x}, \mathbf{z}) + K_2(\mathbf{x}, \mathbf{z})$
2. $K(\mathbf{x}, \mathbf{z}) = a K_1(\mathbf{x}, \mathbf{z})$
3. $K(\mathbf{x}, \mathbf{z}) = K_1(\mathbf{x}, \mathbf{z}) \cdot K_2(\mathbf{x}, \mathbf{z})$
4. $K(\mathbf{x}, \mathbf{z}) = f(\mathbf{x}) \cdot f(\mathbf{z})$
5. $K(\mathbf{x}, \mathbf{z}) = K_3(\Phi(\mathbf{x}), \Phi(\mathbf{z}))$
6. $K(\mathbf{x}, \mathbf{z}) = \mathbf{x}' \mathbf{B} \mathbf{z}$

Pour une preuve, voir [CST00], pp.42-43.

Il est clair que les fonctions noyau encodent des connaissances essentielles sur le problème d'induction étudié, en particulier :

- Une mesure de similarité sur les données.
- La forme fonctionnelle des fonctions de décisions possibles.
- Le type de régularisation réalisé : par exemple les fonctions noyau Gaussiennes pénalisent les dérivées de tous les ordres et favorisent donc les solutions « régulières ».
- Le type de covariance dans l'espace des entrées stipulant comment des points en divers endroits de l'espace sont liés les uns aux autres. On peut ainsi chercher des fonctions noyau invariantes pour certaines transformations de l'espace d'entrée, comme par exemple la rotation.
- Une sorte de distribution de probabilité *a priori* sur l'espace des hypothèses (les fonctions de décision) comme le montre une interprétation bayésienne des SVMs (voir sur ce point le chapitre 16 de [SS02] par exemple).

Il faut donc les choisir avec soin en essayant de traduire grâce à elles le maximum de connaissances préalables dont on dispose sur le problème et sur les données. Le livre [SS02] donne beaucoup d'informations utiles sur ce sujet.

9.3.1.6 La mise en œuvre des SVM

La mise en œuvre de la méthode des séparateurs à vastes marges (SVM) requiert l'accès à un système de résolution de programmation quadratique calculant le vecteur solution \mathbf{x}^* au

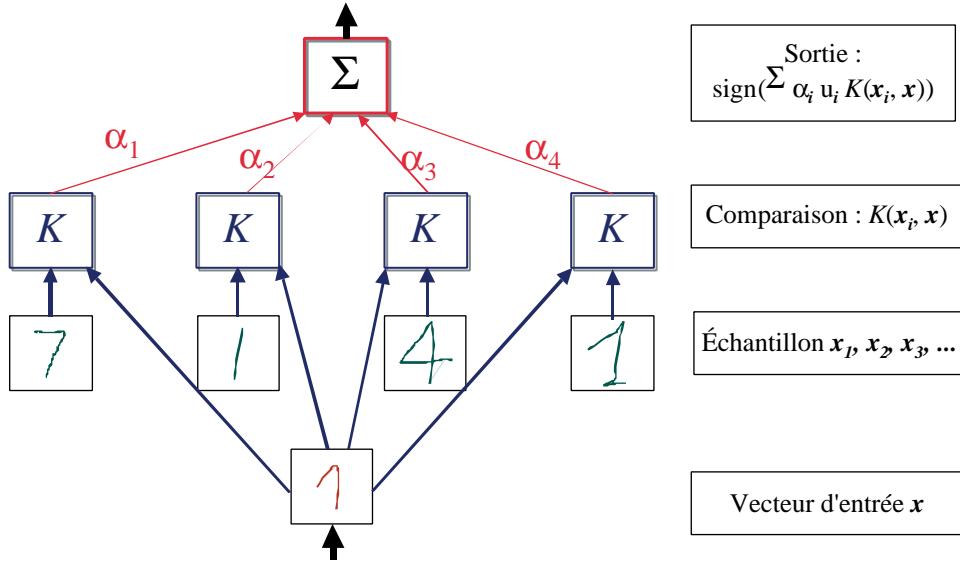


FIG. 9.8 – Cette figure résume le fonctionnement des séparateurs à vastes marges et montre le rôle des fonctions noyau. Lors de l'apprentissage, ici de chiffres manuscrits, un certain nombre d'exemples critiques sont retenus pour définir la fonction de décision. Lorsqu'une nouvelle entrée est présentée au système, elle est comparée aux exemples critiques à l'aide des fonctions noyau qui réalisent un produit scalaire dans l'espace de redescription $\Phi(\mathcal{X})$. La sortie est calculée en faisant une somme pondérée (une combinaison linéaire) de ces comparaisons.

problème suivant :

$$\begin{aligned} \text{Minimiser} \quad & \frac{1}{2} \mathbf{x}' \mathbf{H} \mathbf{x} + \mathbf{c}' \mathbf{x} \\ \text{sous les contraintes} \quad & \begin{cases} \mathbf{A}_1 \mathbf{x} = \mathbf{b}_1 \\ \mathbf{A}_2 \mathbf{x} \leq \mathbf{b}_2 \\ \mathbf{I} \leq \mathbf{x} \leq \mathbf{u} \end{cases} \end{aligned}$$

Les packages permettant de résoudre ce type de problèmes incluent par exemple MINOS, LOQO ou CPLEX (CPLEX Optimization Inc. 1994). Un package accessible est celui d'Alex Smola PR LOQO⁸ avec R qui est une version publique de S-PLUS.

En pratique, les choix de la fonction noyau et de la valeur de la constante C sont souvent faits en utilisant une méthode de validation croisée et en testant l'effet de différents choix. Il est recommandé d'avoir recours à un ensemble de validation pour évaluer la qualité du choix final (voir 3.4 pour les procédures de tests et de validation en apprentissage).

9.3.2 Quelle justification pour les SVM ?

La première partie sur les séparateurs à vastes marges était consacrée à la réalisabilité de la méthode : comment trouver un hyperplan séparateur permettant de discriminer des formes de deux classes. La procédure à mettre en place est finalement relativement simple mais elle repose crucialement sur deux « miracles ». D'une part, il est possible de transformer la formulation

8. Disponible à <http://www.kernel-machines.org/>. avec d'autres implantations.

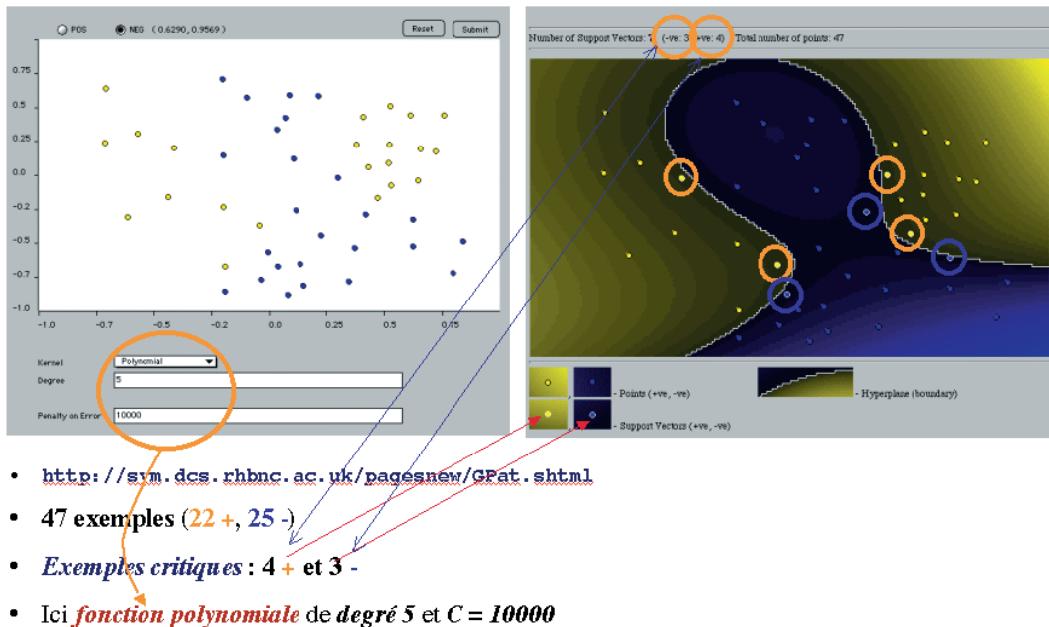


FIG. 9.9 – Dans la fenêtre de gauche, quarante-sept exemples d'apprentissage (vingt-deux de la classe '+’ et vingt-cinq de la classe ‘-’) ont été disposés par l'utilisateur. La fenêtre de droite montre la frontière de décision obtenue avec un SVM de noyau polynomial de degré 5 avec une constante $C = 10000$ (c'est-à-dire avec une faible tolérance aux exemples mal classés). La frontière s'appuie sur sept exemples critiques dont quatre dans la classe '+' et trois dans la classe '-'.

primaire du problème d'optimisation, impossible à résoudre en pratique sauf pour des problèmes jouets, en une formulation duale accessible aux méthodes connues de programmation quadratique D'autre part, cette formulation duale permet d'envisager d'utiliser des espaces de très grandes dimensions dans lesquels une séparation linéaire est possible, grâce à une redescription implicite des données permise par des fonctions noyau.

Il est cependant à craindre qu'un troisième miracle soit nécessaire pour que la méthode des SVM soit utilisable.

La question est en effet celle-ci: lorsque nous utilisons un espace de très grande dimension dans l'espoir d'y trouver une séparatrice linéaire des données, nous avons recours à un espace d'hypothèses très riche, c'est d'ailleurs pour cela que nous sommes confiants de trouver une hypothèse adéquate par rapport aux données (c'est-à-dire de risque empirique faible). Mais alors, l'analyse faite par Vapnik et décrite dans le chapitre 2 ne nous conduit-elle pas à penser que dans ce cas la corrélation entre risque empirique et risque réel est hasardeuse, nullement garantie ? Certes le stratagème du passage par un espace de redescription des données nous aurait permis d'identifier aisément une fonction séparatrice des données, mais nous n'aurions aucune assurance qu'elle se révèle performante sur les futures observations. Cette impression inquiétante est confirmée par le fait que la dimension de Vapnik-Chervonenkis de l'espace des fonctions séparatrices linéaires dans un espace \mathbb{R}^d est égale à $d + 1$, donc proportionnelle à la dimension de l'espace de redescription (qui peut-être infinie). Cette observation semble ruiner tout espoir d'utilisation à bon escient des séparateurs à vastes marges. Alors ?

Si nous réexaminons l'analyse théorique de l'induction, nous observons qu'elle fournit une borne supérieure (en probabilité) sur le risque réel en fonction du risque empirique et de la

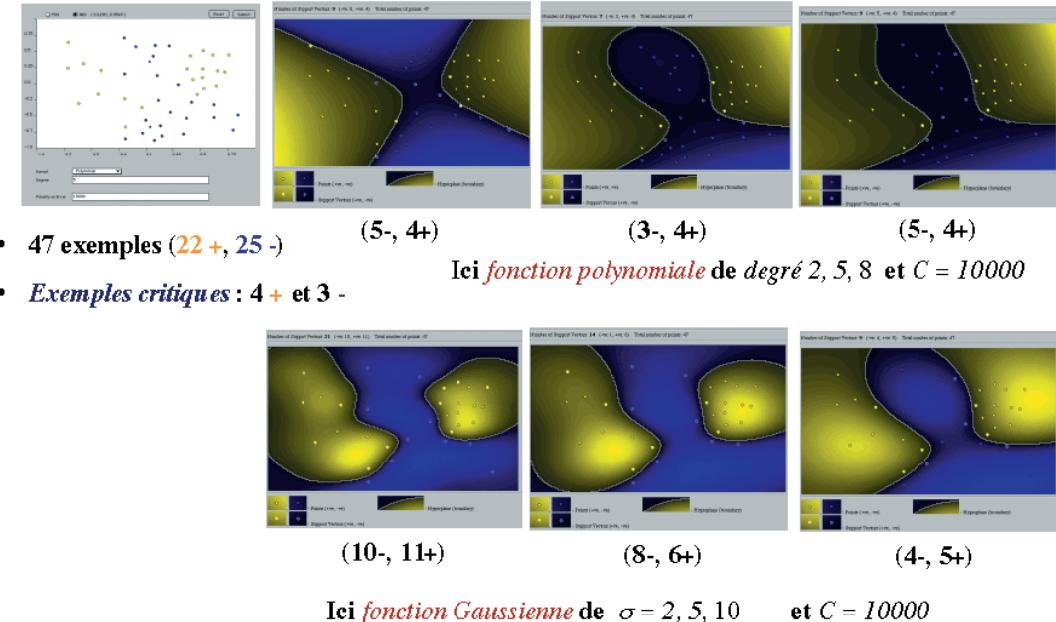


FIG. 9.10 – Pour les mêmes exemples d'apprentissage que ceux de la figure 9.9, on voit l'effet de différents choix de fonctions noyau (polynomiale de degré 2, 5 et 8 en haut et Gaussienne d'écart-type 2, 5 et 10 en bas). Le nombre d'exemples critiques de chaque classe est indiqué entre parenthèses en-dessous des imagettes.

dimension de Vapnik-Chervonenkis d_{VC} . Rappelons par exemple que dans le cas de fonctions indicatrices définies de \mathcal{X} dans $\{-1, 1\}$ avec une fonction de perte 0-1 (comptant les erreurs de prédiction), alors pour toute distribution de probabilité sur $\mathcal{X} \times \{-1, 1\}$, avec une probabilité $1 - \delta$ pour un échantillon de m exemples \mathbf{x} tirés aléatoirement, toute hypothèse $h \in \mathcal{H}$ a son risque réel majoré par :

$$2 R_{Emp}(h) + \frac{1}{m} \left(4 \log\left(\frac{4}{\delta}\right) + 4d_{VC} \log\left(\frac{2em}{d_{VC}}\right) \right) \quad (9.46)$$

du moment que $d_{VC} \leq m$.

Il s'agit d'une borne supérieure, mais il est possible de montrer que cette borne est asymptotiquement serrée puisqu'il existe des distributions pour lesquelles la borne inférieure sur le risque réel est proche de la borne supérieure. Cependant, il faut noter une différence importante entre la borne supérieure, valable pour toute distribution, et la borne inférieure, calculée pour certaines distributions particulièrement défavorables (pour lesquelles les points peuvent être pulvérisés par \mathcal{H}). De fait, l'expérience montre que la borne supérieure est souvent très pessimiste. Il existe donc dans les données correspondant à des problèmes réels des distributions favorables, que l'on appelle aussi *bienveillantes* (*benign distributions*), permettant une corrélation beaucoup plus étroite entre le risque empirique mesuré et le risque réel que ce que prévoit la théorie. Peut-être la technique des séparateurs à vastes marges permettrait-elle de détecter de telles situations et donc d'avoir confiance, quand c'est possible, dans la valeur de l'hypothèse retournée, même si celle-ci résulte du passage dans un espace de redescription de très grande dimension...

9.3.2.1 Une minimisation du risque structurel dépendant des données

Rappelons que le principe de minimisation du risque structurel (voir section 17.2.1 du chapitre 2) cherche à minimiser l'espérance de risque (le risque réel) en contrôlant à la fois la richesse de l'espace d'hypothèses et le risque empirique obtenu avec la meilleure hypothèse possible dans l'espace d'hypothèses disponible. L'idée de Vapnik est alors de définir une structure hiérarchique *a priori* sur les espaces d'hypothèses \mathcal{H}_i en mesurant leur dimension de Vapnik-Chervonenkis. Plus celle-ci est importante, plus l'espace associé est riche et permet de s'adapter à des données diverses.

On peut alors imaginer faire la même chose dans le cas des séparateurs à vastes marges. En effet, plus la marge imposée entre les données est importante, et moins il existe de fonctions séparant les données, c'est-à-dire moins l'espace des fonctions est souple. La marge pourrait alors jouer un rôle analogue à la dimension de Vapnik-Chervonenkis pour caractériser la richesse de l'espace d'hypothèses.

Pour réaliser ce programme (dont nous gommons outrageusement la subtilité technique ici⁹), il est utile de définir une généralisation de la dimension de Vapnik-Chervonenkis pour le cas des fonctions à valeur réelle. Il s'agit de la *fat-shattering dimension* (que l'on peut traduire par « dimension de γ -pulvérisation »).

Définition 9.2 (*Fat-shattering dimension*)

Soit \mathcal{H} un ensemble de fonctions à valeur réelle. On dit qu'un ensemble de points X est γ -pulvérisé par \mathcal{H} s'il existe des nombres réels $r_{\mathbf{x}}$ indexés par $\mathbf{x} \in X$ tels que pour tout vecteur binaire indexé par X (une valeur $b_{\mathbf{x}}$ est attachée à chaque point \mathbf{x}), il existe une fonction $h_b \in \mathcal{H}$ vérifiant :

$$h_b(\mathbf{x}) \begin{cases} \geq r_{\mathbf{x}} + \gamma & \text{si } b_{\mathbf{x}} = 1 \\ \leq r_{\mathbf{x}} - \gamma & \text{sinon} \end{cases} \quad (9.47)$$

En d'autres termes, la dimension de γ -pulvérisation $\text{fat}_{\mathcal{H}}$ de l'ensemble \mathcal{H} est une fonction de \mathbb{R}^+ dans \mathbb{N} qui associe à une valeur de marge γ la taille du plus grand ensemble de points de \mathcal{X} pouvant être γ -pulvérisé, s'il est fini, et la valeur infinie sinon. La dimension de γ -pulvérisation peut être considérée comme la fonction de Vapnik-Chervonenkis des fonctions à seuil mais avec en plus une marge imposée.

Un théorème datant de 1998 (voir [SBE99], p.47) montre que la dimension de γ -pulvérisation permet d'obtenir des bornes d'erreur en généralisation pour les séparateurs à vastes marges, similaires aux bornes d'erreur obtenues par Vapnik pour les fonctions indicatrices. On retrouve en particulier que si l'erreur empirique est nulle, l'erreur en généralisation converge vers 0 au rythme de $1/m$, tandis qu'elle converge au rythme de $1/\sqrt{m}$ lorsqu'elle est non nulle (cas des variables ressort : on admet des erreurs). Cette discontinuité avait déjà été observée pour l'analyse de Vapnik.

La dimension de γ -pulvérisation définit ainsi une sorte de mesure dépendant de l'échelle, liée à la marge, à laquelle on veut étudier les données. Lorsque l'on impose que les données puissent être discriminées avec une vaste marge, la classe des fonctions discriminantes associées peut avoir une complexité (mesurée par la dimension de γ -pulvérisation) bien moindre que la complexité mesurée par la dimension de Vapnik-Chervonenkis, et c'est ce qui explique, grâce aux théorèmes mentionnés, que le risque réel soit corrélé avec le risque empirique mesuré sur les données. Il faut bien voir que la marge mesurée sur l'échantillon d'apprentissage donne une indication de

9. Le lecteur exigeant est invité à se reporter à [SBE99] pp.45-48, ou à [AB96] par exemple.

la chance que l'on a avec la distribution des données. Si elle est grande, nous sommes tombés sur des données pour lesquelles les fonctions séparatrices linéaires sont bien adaptées. Si elle est petite ou inexiste, alors il n'y a plus de corrélation entre risque empirique et risque réel. C'est donc bien seulement pour des distributions « bienveillantes » ou *conciliantes* (révélées par l'existence d'une vaste marge) que l'induction est possible.

Dans cet esprit, il faut noter une différence conceptuelle importante entre le principe de minimisation du risque structurel et l'analyse présentée ici. Le principe SRM prescrit en effet qu'un ordre sur les espaces d'hypothèses \mathcal{H}_i soit *défini a priori*, indépendamment de tout échantillon de données. Au contraire, dans le cas des séparateurs à vastes marges, la hiérarchie sur les espaces d'hypothèses ne peut être définie *qu'a posteriori*, après que les données aient été classées et les marges possibles mesurées avec l'erreur empirique correspondante. D'un certain côté, les données sont utilisées une première fois pour déterminer la hiérarchie des espaces d'hypothèses à employer :

$$\mathcal{H}_\gamma = \{h \mid h \text{ a une marge d'au moins } \gamma \text{ sur les données } \mathcal{S}\}$$

puis, une seconde fois, pour choisir la meilleure hypothèse dans chacun de ces espaces. On commet là une faute à laquelle l'analyse théorique ne peut faire échapper que par une démarche précautionneuse (passant par une majoration de l'erreur qui, elle, est indépendante de la distribution). Ce problème justifie en tout cas l'appellation de principe de minimisation du risque struturel *dépendant des données*.

9.3.2.2 Autres analyses de la méthode SVM

Il existe une autre mesure que la marge permettant d'obtenir une borne sur l'erreur de généralisation obtenue avec les SVM, il s'agit du nombre d'exemples critiques (*support vectors*). Vapnik a montré en 1995 ([Vap95], p.135) que l'espérance du nombre d'exemples critiques permet d'obtenir une borne sur l'espérance de taux d'erreur :

$$E_m(P(\text{erreur})) \leq \frac{E_m[\text{nombre d'exemples critiques}]}{m - 1}$$

où l'opérateur E_m dénote l'espérance mesurée sur un échantillon de taille m . Cette borne est également indépendante de la dimension de l'espace : si on peut trouver un hyperplan optimal qui sépare les deux classes, défini par un petit nombre d'exemples critiques, alors il devrait avoir une bonne performance en généralisation, quelle que soit la dimension de l'espace.

En appliquant une analyse de la performance en généralisation fondée sur la capacité à comprimer l'expression des données (voir section 17.3), Littlestone et Warmuth [LW86] ont également obtenu une borne sur l'erreur réelle de généralisation : l'erreur en généralisation (comptée par le nombre d'erreurs de prédiction) d'un séparateur à vastes marges défini par c exemples critiques parmi un échantillon d'apprentissage de taille m est borné avec une probabilité au moins $1 - \delta$ par :

$$\frac{1}{m - c} \left(c \log_2 \frac{em}{c} + \log_2 \frac{m}{\delta} \right)$$

La proportion d'exemples critiques est ainsi une autre mesure liée à la dimension intrinsèque des données et au caractère conciliant de leur distribution.

Pour terminer ce tour d'horizon sur les analyses théoriques des séparateurs à vastes marges, nous citerons les travaux très intéressants établissant un lien entre les fonctions noyau choisies et la théorie de la régularisation (voir section 17.2.2). Il a ainsi été montré que certaines fonctions noyau correspondaient à certains opérateurs de régularisation. Cela permet à la fois de mieux comprendre le rôle de ces fonctions noyau, mais aussi d'envisager de nouvelles méthodes d'apprentissage généralisant l'approche des SVM (voir par exemple [SBE99], p.14).

9.3.2.3 SVM et Bayes Point Machines

Récemment est apparu le concept de *séparateur linéaire de Bayes* (*Bayes Point Machines*) (voir par exemple [Her02]). L'idée est la suivante: d'après l'approche bayésienne de l'induction (voir section 2.4), la meilleure décision y à prendre en un point \mathbf{x} est la moyenne pondérée par leur probabilité *a posteriori* des décisions de chaque hypothèse:

$$y = \text{Bayes}_S(\mathbf{x}) = \int_{\mathcal{H}} p_{\mathcal{H}}(h|\mathcal{S}) h(\mathbf{x}) dp_{\mathcal{H}} \quad (9.48)$$

ou encore, si l'on considère la fonction de perte l :

$$y = \text{Bayes}_S(\mathbf{x}) = \underset{y}{\operatorname{Argmin}} \int p_{\mathcal{H}}(h|\mathcal{S}) l(h(\mathbf{x}), y) dp_{\mathcal{H}} \quad (9.49)$$

Nous avons vu que cette règle de décision minimise l'espérance de risque lorsque l'on dispose d'une mesure de probabilité sur les hypothèses de \mathcal{H} . Une propriété de cette règle de décision est qu'elle peut réaliser une fonction de décision qui est en dehors de l'espace des hypothèses \mathcal{H} (dans le cas où celui-ci n'est pas convexe). Si cette règle de décision est optimale en moyenne, elle est très difficile à mettre en œuvre car elle implique pour chaque exemple \mathbf{x} de calculer *toutes* les décisions possibles $h(\mathbf{x})$ prises sur \mathcal{H} pondérées par leur probabilité *a posteriori*. Il est cependant possible de limiter cette difficulté en considérant des espaces d'hypothèses \mathcal{H} pour lesquels la décision $h(\mathbf{x})$ est facile à déterminer. C'est particulièrement le cas si l'on considère l'espace des séparateurs linéaires.

Comme l'espace des séparateurs linéaires est convexe (toute combinaison linéaire de séparateurs linéaires est un séparateur linéaire), la règle de décision bayésienne pour déterminer la réponse y en un point \mathbf{x} revient à identifier un séparateur linéaire.

Tout séparateur linéaire est caractérisé par son vecteur de poids \mathbf{w} . Lorsque l'on cherche un séparateur d'erreur nulle sur l'échantillon d'apprentissage (hypothèse cohérente avec l'échantillon), n'importe quelle hypothèse prise dans l'espace des versions est satisfaisante. Dans le cas des séparateurs à vastes marges, l'hypothèse retenue est celle qui est maximalement distante des contraintes posées par les données, ce qui est schématisé par la croix dans la figure 9.11. Le séparateur linéaire de Bayes choisit quant à lui le séparateur dont le vecteur poids $\mathbf{w}_{\text{Bayes}}$ est au centre de masse de l'espace des versions (figuré par un rond dans la figure 9.11).

Il peut être montré que le séparateur associé est asymptotiquement proche de la décision bayésienne optimale. Il n'est cependant pas aisément de trouver ce centre de masse en général, et des techniques de parcours de rayons par billard sont actuellement à la mode pour résoudre ce problème, mais sans garanties théoriques.

9.3.3 La régression par fonctions noyau et exemples critiques

Même si la méthode des séparateurs à vastes marges est spécifique aux problèmes de classification, on peut en étendre le champ au problème de la régression, c'est-à-dire de la recherche d'une fonction $h(\mathbf{x}) = y$ dans \mathbb{R} telle que pour tous les points d'apprentissage $\{(\mathbf{x}_i, u_i)\}_{1 \leq i \leq m}$ $f(\mathbf{x}_i)$ soit le plus « proche » possible de u_i . Vapnik propose d'utiliser la fonction de perte suivante:

$$|y - f(\mathbf{x})|_{\varepsilon} := \max\{0, |y - f(\mathbf{x})| - \varepsilon\} \quad (9.50)$$

Alors, afin d'estimer la régression linéaire:

$$f(\mathbf{x}) := (\mathbf{w} \cdot \mathbf{x}) + b \quad (9.51)$$

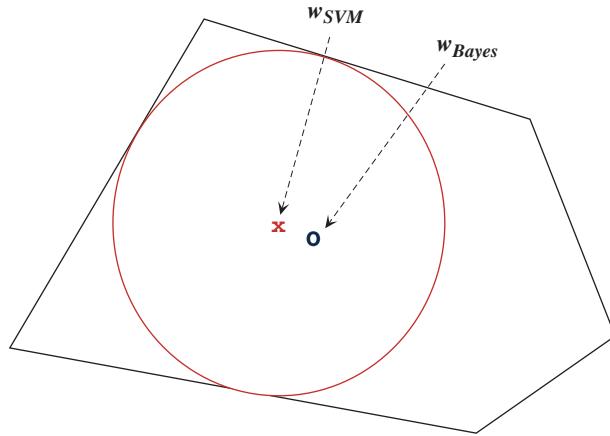


FIG. 9.11 – Dans l'espace des hypothèses (l'espace des vecteurs poids \mathbf{w}), l'espace des versions est délimité par des droites correspondant aux contraintes posées par les exemples d'apprentissage. La méthode des séparateurs à vastes marges retourne le vecteur de poids (figuré par une croix) le plus éloigné de toutes les contraintes. La méthode des séparateurs linéaires de Bayes retourne le vecteur de poids correspondant au centre de masse de l'espace des versions (figuré ici par un cercle).

avec une précision de ε , on minimise :

$$\frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^m |y_i - f(\mathbf{x}_i)|_\varepsilon \quad (9.52)$$

On peut réexprimer ce problème sous forme d'un problème d'optimisation sous contraintes (voir [Vap95]) :

$$\begin{aligned} \text{Minimiser} \quad & \tau(\mathbf{w}, \xi, \xi^*) = \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{sous les contraintes} \quad & \begin{cases} ((\mathbf{w} \cdot \mathbf{x}_i) + b) - y_i \leq \varepsilon + \xi_i \\ y_i - ((\mathbf{w} \cdot \mathbf{x}_i) + b) \leq \varepsilon + \xi_i \\ \xi_i, \xi_i^* \geq 0 \end{cases} \end{aligned} \quad (9.53)$$

pour tous les $i = 1, \dots, m$. On peut noter à nouveau que toute erreur plus petite que ε ne requiert pas une valeur non nulle de ξ ou de ξ_i et donc ne doit pas être prise en compte par la fonction objectif 9.53. La figure 9.12 illustre le rôle des contraintes.

Comme dans le cas de la classification, on peut généraliser à la régression non linéaire en passant par un espace de redescription des entrées grâce à l'utilisation de fonctions noyau. L'introduction de multiplicateurs de Lagrange conduit au problème d'optimisation suivant, dans lequel les constantes $C > 0$ et $\xi \geq 0$ sont choisies *a priori*:

$$\begin{aligned} \text{Maximiser} \quad & W(\boldsymbol{\alpha}, \boldsymbol{\alpha}) = -\varepsilon \sum_{i=1}^m (\alpha_i^* + \alpha_i) + \sum_{i=1}^m (\alpha_i^* - \alpha_i) y_i \\ \text{sous les contraintes} \quad & \begin{cases} 0 \leq \alpha_i, \alpha_i^* \leq C \quad i = 1, \dots, m \\ \sum_{i=1}^m (\alpha_i - \alpha_i^*) = 0 \end{cases} \end{aligned} \quad (9.54)$$

L'estimation de la régression prend alors la forme:

$$f(\mathbf{x}) = \sum_{i=1}^m (\alpha_i^* - \alpha_i) k(\mathbf{x}_i, \mathbf{x}) + b \quad (9.55)$$

où b est calculé en utilisant le fait que la contrainte $((\mathbf{w} \cdot \mathbf{x}_i) + b) - y_i \leq \varepsilon + \xi_i$ devient une égalité avec $\xi_i = 0$ si $0 < \alpha_i < C$, et la contrainte $y_i - ((\mathbf{w} \cdot \mathbf{x}_i) + b) \leq \varepsilon + \xi_i$ devient une égalité avec $\xi_i^* = 0$ si $0 < \alpha_i^* < C$.

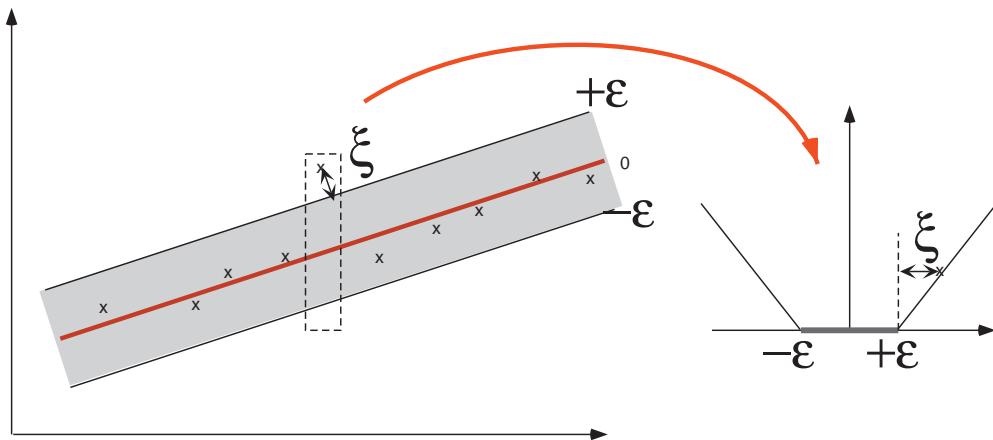


FIG. 9.12 – Dans la régression par SVM, au lieu d'imposer une marge entre les points des deux classes, on impose une sorte de « chaussette » autour des points grâce au paramètre ε . Le compromis entre la complexité du modèle (la fonction de régression) et la fidélité aux points d'apprentissage est réglée par la variable ressort ξ .

9.3.4 Conclusions sur les SVM

La méthode des séparateurs à vastes marges est, avec la méthode du *boosting* (voir chapitre 11), une des rares méthodes d'apprentissage qui soient complètement issues de considérations théoriques. De ce fait, elle est bien fondée mathématiquement et l'on en connaît un certain nombre de propriétés. En particulier ces méthodes règlent automatiquement le compromis entre la complexité de la classe d'hypothèses et la nécessaire fidélité aux données. Par ailleurs, l'approfondissement de leur justification théorique a conduit à des raffinements des principes induktifs en prenant en compte, en plus de la fidélité aux données et de la complexité de la classe d'hypothèses, la distribution des données d'apprentissage. La compréhension complète de cette nouvelle caractéristique demandera encore des recherches.

Une autre conséquence des travaux sur les SVM est la réalisation que bien d'autres méthodes peuvent bénéficier de l'emploi de fonctions noyau. Celles-ci correspondent en effet à de nouvelles mesures de distance ou de corrélation applicables dans des espaces beaucoup plus généraux que ce qui était envisagé dans les méthodes traditionnelles.

Pour ces raisons d'ordre conceptuel et parce que les SVM, qui sont faciles à mettre en œuvre, donnent souvent de bons résultats en apprentissage, il y a une grande excitation autour des méthodes que l'on appelle maintenant méthodes d'apprentissage par fonctions noyau (*kernel-based machine learning*). Il s'agit là d'un domaine dont on peut attendre beaucoup de développements et d'applications intéressantes.

Notes historiques et approfondissements

La théorie de la discrimination linéaire remonte aux années 1930, lorsque Fisher proposa une procédure de classification. Dans le domaine de l'intelligence artificielle, l'attention fut attirée vers cette question par les travaux de Rosenblatt qui commença à étudier la règle d'apprentissage du perceptron à partir de 1956. Minsky et Papert, dans leur célèbre livre [MP69], analysèrent les limitations des machines linéaires. Le livre de Duda, Hart et Stork [DHS01] offre un panorama très complet des recherches sur ces machines.

L'idée d'hyperplans à marge maximale a été redécouverte plusieurs fois. Elle a été discutée par Vapnik et Lerner dès 1963 [VL63], par Duda et Hart dans leur premier livre de 1973, tandis que les chercheurs étudiant l'apprentissage artificiel sous l'angle de la physique statistique (voir le chapitre 17) en voyait également l'avantage [EdB01] et proposaient le modèle de l'*Adatron* [AB89].

Ce n'est cependant qu'en 1992 que tous les ingrédients des séparateurs à vastes marges furent rassemblés par Vapnik et des collègues, et c'est seulement en 1995 qu'apparut le concept de variables ressort et de marge souple et que les SVM devinrent connus dans la communauté, grâce surtout au livre de Vapnik [Vap95]. L'utilisation et l'analyse subséquente des SVM a relancé l'intérêt pour toutes les approches fondées sur les fonctions noyau (*kernel-based methods* [SBE99, SS02] (voir aussi le chapitre 14).

L'idée de relier le concept de marge à celui de la dimension de pulvérisation (*fat-shattering dimension*) est apparue implicitement dans plusieurs références, mais fut introduite explicitement dans [KS94], tandis que [STBWA98] rendait populaire la notion de borne sur le risque réel dépendant de la distribution des données (d'où le terme de *luckiness* choisi par les auteurs pour souligner l'idée de profiter si c'est possible d'une corrélation entre l'espace d'hypothèses et la distribution des données). Cela stimula de nouvelles approches divergeant de l'approche de Vapnik, laquelle ne prend pas en compte la distribution des données.

Résumé

Ce chapitre a montré comment on peut apprendre des fonctions de décision linéaires dans l'espace des entrées. Dans le cas de la classification binaire, cela revient à couper l'espace en deux par un hyperplan.

L'intérêt de ces fonctions de décision est qu'elles sont simples à apprendre (correspondant souvent à un problème d'optimisation quadratique, donc à un seul optimum), et qu'il est facile de caractériser le meilleur séparateur, contrairement à des séparations non linéaires.

Les méthodes classiques d'apprentissage par itération ont été décrites.

L'une des révolutions de ces dernières années en apprentissage concerne des méthodes motivées par des considérations théoriques sur l'induction et qui se traduisent par la recherche de séparateurs linéaires dans lesquels on cherche une marge maximale avec les exemples : les séparateurs à vaste marge. En utilisant des fonctions noyau qui permettent une redescription des exemples dans un espace de plus grande dimension, on peut étendre le champ de ces méthodes bien fondées à des séparatrices non linéaires dans l'espace d'entrée. Il s'agit donc là d'une approche très prometteuse et qui suscite beaucoup de travaux.

Chapitre 10

L'apprentissage de réseaux connexionnistes

Dans ce chapitre, nous présentons une technique d'apprentissage fondée au départ sur une analogie avec la physiologie de la transmission de l'information et de l'apprentissage dans les systèmes cérébraux : les modèles connexionnistes (on dit aussi les réseaux de neurones artificiels). Le but du développement de cette approche était à l'origine de modéliser le fonctionnement du cerveau ; cela reste un des axes de recherche du domaine, mais ici nous traiterons seulement de l'application de certains modèles informatiques élémentaires à l'apprentissage automatique de règles de classification.

Le cerveau est un organe caractérisé par l'interconnexion d'un nombre élevé d'unités de traitement simples, les cellules nerveuses ou neurones. Le comportement de ce réseau naturel de neurones est déterminé par son architecture, c'est-à-dire le nombre des cellules et la manière dont elles sont connectées, ainsi que par les poids affectés à chacune des connexions. Chaque connexion entre deux neurones est caractérisée par son poids qui mesure le degré d'influence du premier neurone vers le second. La capacité d'apprentissage de ces systèmes est reliée à la mise en mémoire de nouvelles connaissances par la modification des poids des connexions à partir d'exemples. Pour donner un ordre de grandeur de sa complexité, le cerveau humain comporte environ cent milliards de neurones, chacun relié en moyenne à dix mille autres.

Nous abandonnons maintenant toute forme de référence biologique pour nous intéresser aux réseaux de neurones artificiels, ou réseaux connexionnistes, et en particulier à leur application à l'apprentissage automatique.

Le calcul par réseaux connexionnistes est fondé sur la propagation d'informations entre des unités élémentaires de calcul. Les possibilités de chacune sont faibles, mais leur interconnexion permet d'effectuer un calcul global complexe. Du point de vue de l'apprentissage, les poids des connexions entre ces unités peuvent être réglés sur des ensembles d'exemples : le réseau ainsi entraîné pourra réaliser des tâches de classification ou de régression.

LES EIDERS (*Somatiera mollissima*) sont des canards marins pour lesquels les guides ornithologiques¹ sont pour une fois assez péremptoires : « Le mâle est le seul canard qui paraisse blanc quand on le voit de face et noir quand il est vu de dos (...) Le plumage de la femelle est brun. » Supposons qu'une bande d'eiders vogue à quelque distance des côtes. Comment un avimiteur peut-il distinguer un eider mâle d'un eider femelle uniquement sur la couleur² ?

Le problème est en apparence assez simple. Il faut définir une échelle qui va du blanc au noir et y définir trois zones. On aura une décision du type de celle de la figure 10.1.

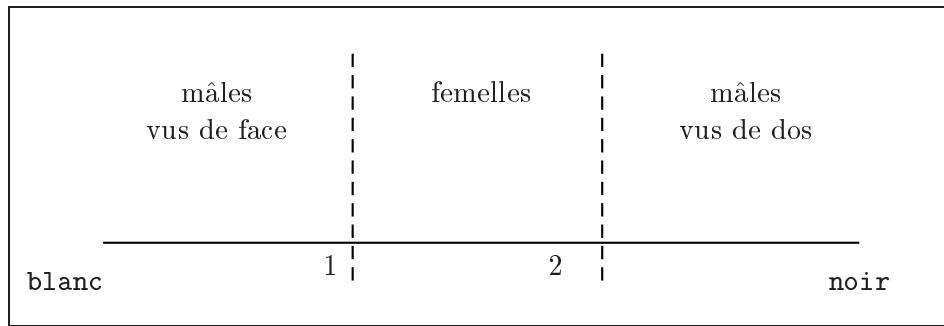


FIG. 10.1 – Répartition des eiders sur l'axe x : blanc / noir.

Notons x le niveau de gris d'un eider dont on cherche à déterminer le sexe. Les connaissances sont les suivantes :

- Si $x \leq 1$ ou $x \geq 2$ alors mâle.
- Si $1 \leq x \leq 2$ alors femelle.

Le concept « sexe de l'animal observé » ne peut donc pas se traduire directement par une seule comparaison.

Les techniques des surfaces séparatrices linéaires que nous avons vues au chapitre précédent sont-elles valables ici ? Non, car ce concept est impossible à décrire de cette façon. En effet, une décision linéaire dans un espace à une seule dimension (ici, la couleur x est le seul attribut du problème) revient à comparer l'attribut à un seuil. Mais ici, deux seuils interviennent et non pas un seul. Il va falloir trouver une technique de décision plus élaborée.

Nous allons décomposer le problème en deux étages de décision linéaire. Le premier produira deux valeurs binaires notées y_1 et y_2 . La première indique si oui ou non x est inférieur au seuil de valeur 2, la seconde si x est supérieur au seuil de valeur 1. Le deuxième étage combinera ces deux valeurs binaires pour décider si le concept est satisfait ou non. C'est ce que montrent le tableau ci-dessous et la figure 10.2.

	$x \leq 1$	$1 \leq x \leq 2$	$x \geq 2$
y_1	VRAI	VRAI	FAUX
y_2	FAUX	VRAI	VRAI

1. H. Heintzel, R. Fitter and J. Parslow *Oiseaux d'Europe*. Delachaux et Niestlé, 1972.

2. On suppose bien sûr qu'il n'y a que des eiders dans la zone observée et qu'aucun mâle ne se présente de profil.

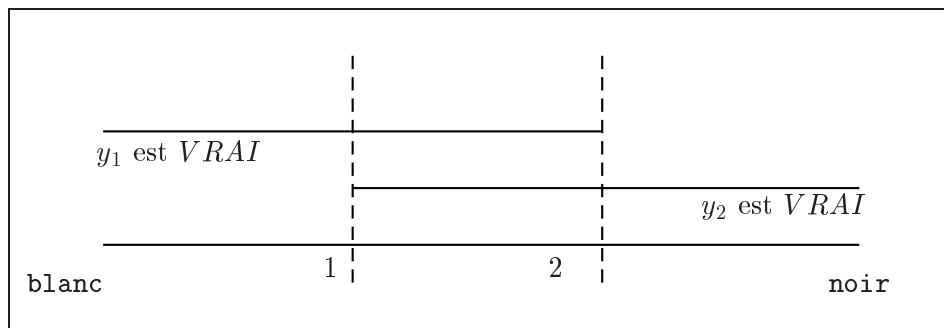


FIG. 10.2 – Deux décisions élémentaires à combiner.

À partir de y_1 et y_2 , la décision est donc la suivante :

y_1	y_2	Concept
VRAI	VRAI	Femelle
VRAI	FAUX	Mâle
FAUX	VRAI	Mâle
FAUX	FAUX	(impossible)

Ce problème a été présenté jusqu'ici dans le cadre de la logique, mais il est facile de le transformer en une décision numérique. Il suffit pour cela d'attribuer les valeurs réelles 0 et 1 à y_1 et y_2 et au concept selon qu'ils sont *VRAI* ou *FAUX*. Il est en effet peu intuitif, mais immédiat de vérifier que la valeur numérique z

$$z = y_1 + y_2 - 1.5$$

est positive quand l'eider est femelle (elle vaut $z = 1 + 1 - 1.5 = 0.5$) et négative quand il est mâle (elle vaut $z = 1 + 0 - 1.5 = -0.5$ ou $z = 0 + 1 - 1.5 = -0.5$). Donc, le concept est aussi exprimé par le signe de la valeur z .

Le concept appris est par conséquent ramené à une décision par surface séparatrice linéaire, non pas dans l'espace de représentation de départ, réduit à x , mais sur deux valeurs binaires y_1 et y_2 , extraites elles-mêmes de décisions linéaires sur x .

En quoi cette technique de décision est-elle conforme aux premières notions que nous avons données sur les réseaux connexionnistes ? Il suffit pour le voir de se placer dans une représentation graphique (figure 10.3). L'information se propage de la gauche vers la droite. Les deux cercles centraux ont pour valeur de sortie y_1 et y_2 , celui de droite a pour valeur de sortie z ; il exprime donc le concept cherché. Cette représentation sera bien sûr plus longuement expliquée dans ce chapitre.

10.1 Les différents éléments d'un réseau connexionniste

Détaillons d'abord les différentes notions nécessaires à la compréhension des réseaux connexionnistes, en particulier des réseaux *multicouches* que nous étudions dans ce chapitre.

L'espace de représentation

Les données d'entrée sont des vecteurs de \mathbb{R}^d , notés comme d'habitude (en transposition) $\mathbf{x}^T = (x_1, \dots, x_d)$. Les réseaux connexionnistes que nous présentons dans ce chapitre sont donc des règles de classification de données numériques (ou logiques).

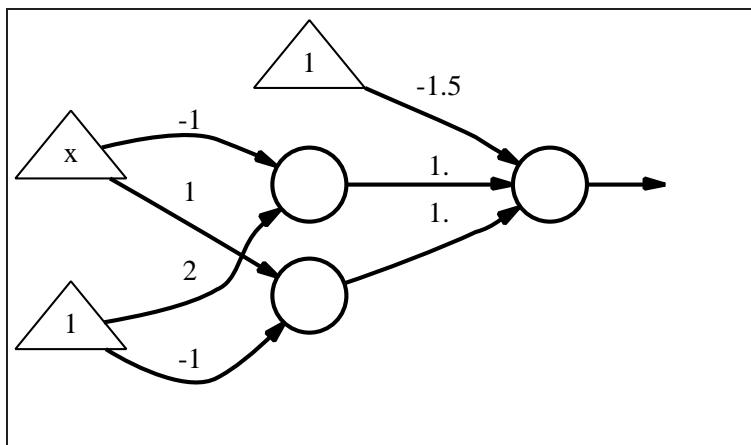


FIG. 10.3 – Un réseau connexionniste qui distingue les eiders mâles des eiders femelles.

Le neurone formel

L’unité de traitement élémentaire dans un réseau connexionniste est capable de faire seulement certaines opérations simples. Ces unités sont souvent appelées *neurones formels* pour leur similitude grossière avec les neurones du cerveau. Les modèles de réseaux connexionnistes qui nous intéressent particulièrement, les *réseaux multicouches* classent les unités selon qu’elles sont des neurones d’entrée, cachés, ou de sortie.

- Un *neurone d’entrée* ou, simplement, une *entrée*, est une unité chargée de transmettre une composante du vecteur \mathbf{x} des données (en particulier, les données d’apprentissage pendant la phase d’apprentissage).
- Un *neurone de sortie* est une unité qui fournit une hypothèse d’apprentissage, par exemple dans un problème de classification, une décision sur la classe à laquelle est attribué \mathbf{x} .
- Enfin, un *neurone caché* est un neurone qui n’est ni un neurone d’entrée, ni un neurone de sortie.

Il existe d’autres modèles, par exemple la *machine de Boltzmann* pour laquelle tous les neurones formels, y compris d’entrée et de sortie, sont connectés les uns aux autres.

L’état d’un neurone formel

Il est commode de décrire un réseau connexionniste à un moment de son fonctionnement par un ensemble de valeurs σ_i , une pour chaque neurone formel i . Lorsque le neurone i est un neurone d’entrée, on a : $\sigma_i = x_i$. Dans tous les autres cas, σ_i est l’état du neurone i , calculé par la règle de propagation décrite ci-dessous au paragraphe 10.2.1.

Comment fonctionne un neurone formel

Un neurone formel est caractérisé par une *fonction de sortie* f qui permet de calculer pour chaque neurone i une valeur de sortie y_i en fonction de son état d’activation σ_i :

$$y_i = f(\sigma_i) \quad (10.1)$$

On peut envisager plusieurs sortes de fonctions de sortie, mais le plus souvent on utilise soit

la fonction signe (comme dans l'exemple d'introduction), soit une fonction sigmoïde³ d'équation

$$y_i = f(\sigma_i) = \frac{1}{1 + e^{-\lambda\sigma_i}}$$

La figure 10.4 montre le graphe de cette fonction pour la valeur $\lambda = 1$.

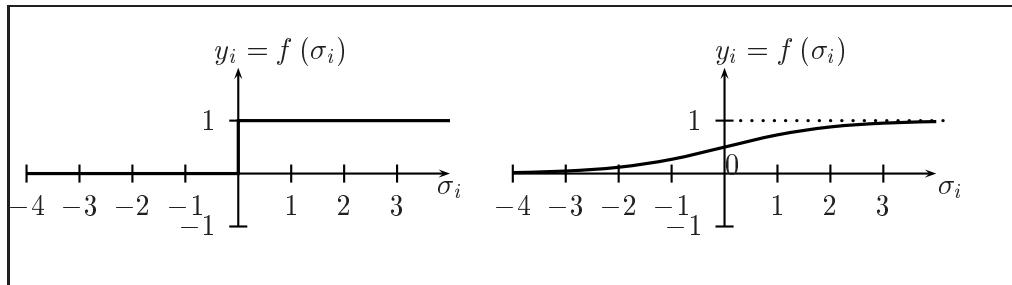


FIG. 10.4 – Deux fonctions non-linéaires de sortie utilisées dans les réseaux connexionnistes. La première est la fonction seuil : elle vaut 0 quand σ_i est négatif et 1 s'il est positif. La seconde est la fonction sigmoïde d'équation $y_i = f(\sigma_i) = \frac{1}{1+e^{-\lambda\sigma_i}}$. Cette fonction est paramétrée par sa pente à l'origine λ . Pour λ très grand, on retrouve la fonction seuil. Pour λ très petit, cette fonction est pratiquement linéaire dans une vaste région autour de l'origine. La sigmoïde de cette figure est dessinée pour $\lambda = 1$.

10.2 L'architecture multicouche

Un réseau est caractérisé par son architecture, c'est-à-dire la structure selon laquelle les neurones formels qui le composent sont reliés les uns aux autres. Certains réseaux, comme les machines de Boltzmann, ont une connectivité complète (chaque neurone formel est relié à toutes les autres unités) ; d'autres, ceux dont on va parler dans ce chapitre, ont une architecture en couches superposées. La caractéristique de ces réseaux est que les unités d'une couche sont reliées à toutes celles de la couche supérieure, mais à aucune autre. À chaque lien entre deux unités i et j , on associe un poids correspondant à la force de la connexion entre ces deux unités, noté $w(i, j)$.

Dans ces modèles, la couche d'entrée sert à la lecture des données et la couche de sortie à traduire la décision. En général, il s'agit d'une décision de classification.

10.2.1 La transmission de l'information dans un réseau multicouche

Le fonctionnement d'un réseau connexioniste, pour des poids de connexions donnés, se résume à définir une *règle de propagation*, qui décrit comment calculer l'état d'activation d'une unité j en fonction des unités i pour lesquelles il existe un poids $w(i, j)$. Appelons $source(j)$ l'ensemble de ces unités i . Le schéma est donné à la figure 10.5

La règle la plus souvent utilisée consiste à calculer la somme des valeurs de sortie y_i des unités $i \in source(j)$, pondérées par les poids des connexions correspondantes.

$$\sigma_j = \sum_{i \in source(j)} w(i, j) y_i$$

3. C'est-à-dire en forme de s.

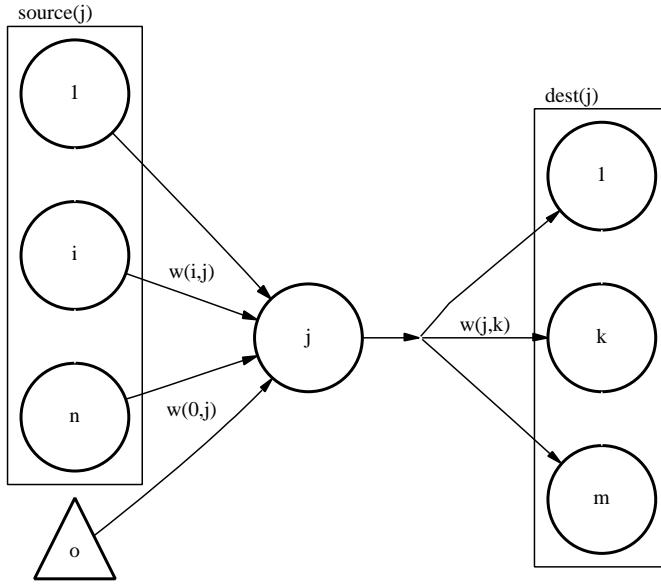


FIG. 10.5 – Le neurone formel: notation des connexions.

Cette règle n'est pas utilisée pour calculer l'état d'activation des neurones d'entrée, puisque leur rôle est simplement de transmettre les entrées. Dans leur cas, on a donc simplement $\sigma_j = x_j$.

De plus, on doit ajouter au vecteur d'entrée une composante supplémentaire, appelée *offset*⁴, représentée par un triangle dans la figure 10.5 et dont la valeur est le plus souvent fixée arbitrairement à 1. Bien que ce ne soit pas nécessaire, une entrée du même type est souvent rajoutée à chaque couche cachée, pour des raisons d'homogénéité dans les algorithmes de reconnaissance et d'apprentissage. Pour chaque couche, on ajoute donc un neurone formel dans lequel n'arrive aucune connexion, dont l'activité est toujours égale à 1 et dont les transitions vers les neurones formels j de la couche supérieure sont notées $w(0, j)$.

L'équation de fonctionnement de chaque neurone j est finalement la suivante :

$$\sigma_j = w(0, j) + \sum_{i \in \text{source}(j)} w(i, j) y_i \quad (10.2)$$

En résumé, dans le cas d'un *réseau à couches*, la couche d'entrée est activée par l'arrivée d'une donnée, en recevant une composante du vecteur \mathbf{x} sur chacune de ses unités. La première couche cachée effectue le calcul ci-dessus (équation 10.2) pour chacune de ses unités, puis c'est au tour de la seconde, etc. Finalement, l'unité de la couche de sortie ayant la valeur la plus forte indique la classe calculée pour l'entrée. Un réseau connexionniste multicouche général est représenté sur la figure 10.6.

10.2.2 Un exemple

Considérons le réseau à deux entrées x_1 et x_2 de la figure 10.7. Il possède une couche cachée composée des neurones formels numérotés 3 et 4. Sa couche de sortie est composée d'un seul neurone formel, numéroté 5. Il y a une valeur d'offset fixée à 1 pour le vecteur d'entrée et une autre pour la couche cachée.

4. Le mot français serait malheureusement « biais », qui a un autre sens en apprentissage.

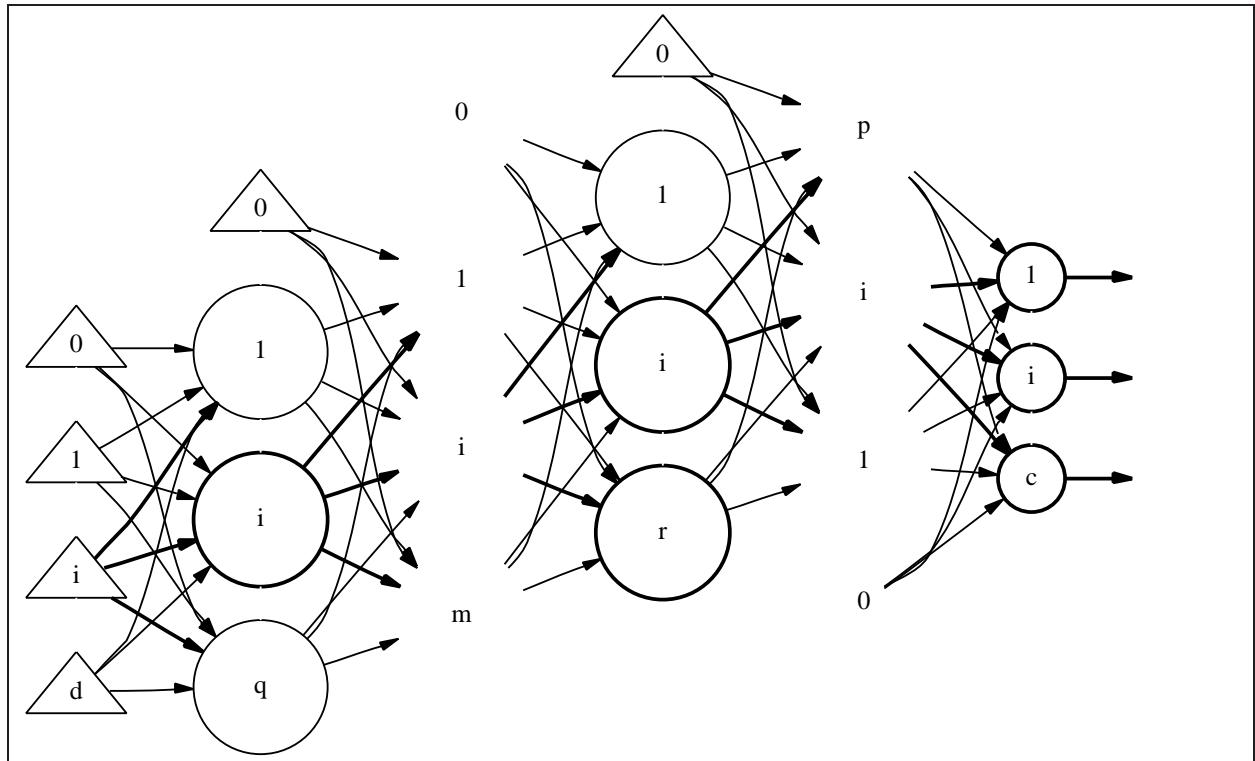


FIG. 10.6 – Le réseau multicouche général. Dans ce graphisme, l'information se propage de la gauche vers la droite. Les neurones formels sont indiqués par des cercles. Les composantes du vecteur d'entrée et les offsets sont à l'intérieur des triangles.

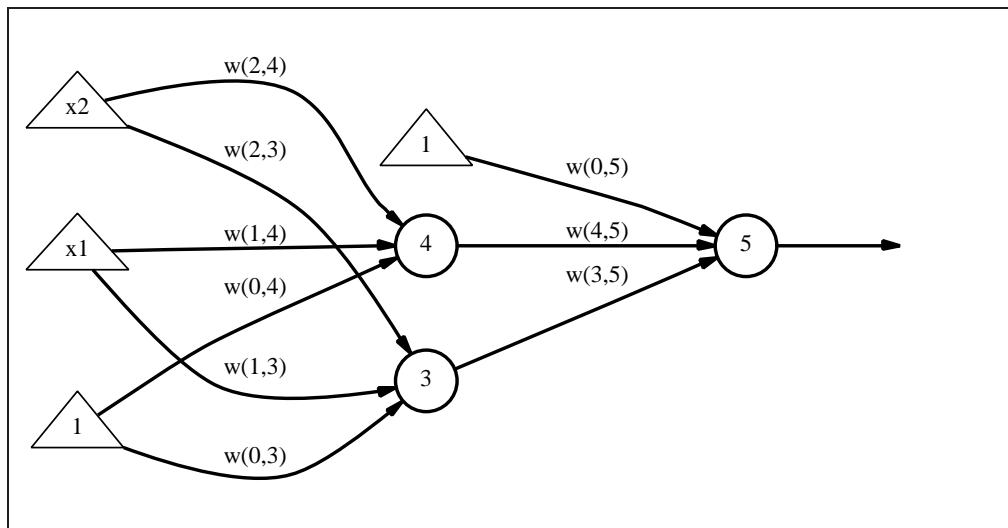


FIG. 10.7 – Un exemple de réseau multicouche : la notation des neurones formels et des poids des connexions.

Fixons maintenant (figure 10.8) la valeur des poids comme suit :

$$\begin{aligned}
 w(0, 3) &= 0.2 & w(1, 3) &= 0.1 & w(2, 3) &= 0.3 \\
 w(0, 4) &= -0.3 & w(1, 4) &= -0.2 & w(2, 4) &= 0.4 \\
 w(0, 5) &= 0.4 & w(3, 5) &= 0.5 & w(4, 5) &= -0.4
 \end{aligned}$$

et prenons pour vecteur d'entrée $\mathbf{x} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$

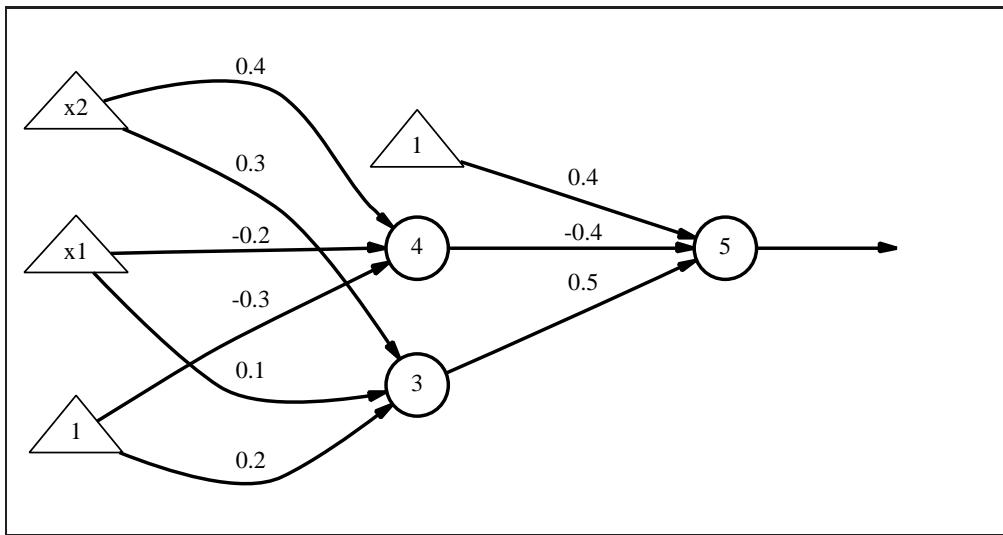


FIG. 10.8 – Le même exemple avec des valeurs numériques pour les poids des connexions.

La propagation des calculs s'effectue alors comme indiqué dans la table ci-dessous :

Neurone formel j		σ_j	y_j
3	$0.2 + 0.1 \times 1 + 0.3 \times 1 = 0.6$	$\frac{1}{1+e^{-0.6}} \simeq 0.65$	
4	$-0.3 + -0.2 \times 1 + 0.4 \times 1 = -0.1$	$\frac{1}{1+e^{0.1}} \simeq 0.48$	
5	$0.4 + 0.5 \times 0.65 - 0.4 \times 0.48 = 0.53$	$\frac{1}{1+e^{-0.53}} \simeq 0.63$	

10.2.3 Un autre exemple : le problème « XOR »

Cet exemple a pour but de montrer comment l'introduction de la fonction non linéaire de sortie des neurones formels et de l'architecture en couches permet d'obtenir des surfaces séparatrices non linéaires.

Plaçons nous dans \mathbb{R}^2 , avec quatre points d'apprentissage situés au quatre coins du carré unité. Chaque paire de points opposés en diagonale forme une classe. Les exemples ont donc la forme suivante :

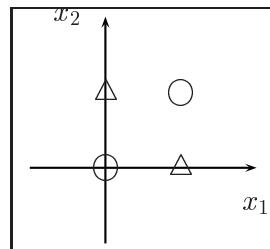


FIG. 10.9 – Le problème XOR : les deux points Δ sont des exemples de la même classe, les deux points \circ des exemples d'une autre classe. Les deux classes ne sont pas linéairement séparables.

Le réseau connexionniste de la figure 10.10 permet de résoudre le problème. En choisissant $y_i = f(\sigma_i)$ comme la fonction seuil, la propagation des calculs se fait comme indiqué dans le tableau associé.

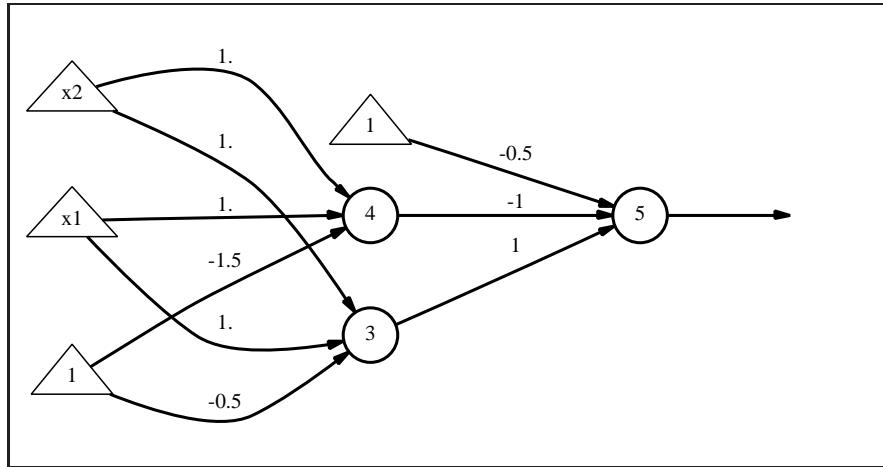


FIG. 10.10 – Un réseau « XOR ».

x_1	x_2	σ_3	y_3	σ_4	y_4	σ_5	y_5
0	0	-0.5	0	-1.5	0	-0.5	0
0	1	0.5	1	-0.5	0	0.5	1
1	0	0.5	1	-0.5	0	0.5	1
1	1	1.5	1	0.5	1	-0.5	0

Un interprétation est possible en logique booléenne: au lieu de considérer les valeurs 0 et 1 comme des coordonnées numériques, prenons-les comme des valeurs logiques. Dans ce cas, on peut interpréter les sorties intermédiaires et la sortie finale comme des fonctions booléennes sur les entrées. Le réseau réalise la fonction *XOR* (le *OU exclusif*), qui vaut 0 pour les deux points Δ et 1 pour les deux points \bigcirc . Ceci est rendu possible par les non-linéarités du système de calcul (voir la figure 10.11).

$$\begin{aligned} y_3 &= x_1 \vee x_2 \\ y_4 &= x_1 \wedge x_2 \\ y_5 &= y_3 \wedge \neg y_2 = x_1 \text{ } XOR \text{ } x_2 \end{aligned}$$

10.2.4 Le protocole d'apprentissage

Dans le cas des réseaux connexionnistes, les données d'apprentissage sont en général présentées séquentiellement ; l'apprentissage est donc incrémental. Chaque étape emploie une donnée pour modifier les poids des connexions. La suite des données utilisées peut être construite par un tirage aléatoire avec remise dans l'ensemble des exemples ou par plusieurs passages successifs de la totalité de cet ensemble. Au total, le nombre de données utilisées pour l'apprentissage est en général bien supérieur au nombre d'exemples : chacun est utilisé en moyenne ou exactement un grand nombre de fois (couramment une centaine de fois).

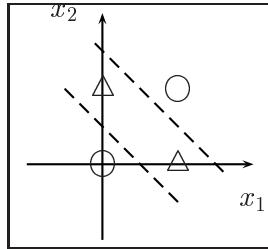


FIG. 10.11 – Une des façons de résoudre le problème *XOR* avec un réseau connexionniste à une couche cachée. La zone affectée à la classe \triangle est le « couloir » compris entre les deux droites en pointillé, celle affectée à la classe \circ est à l’extérieur. La première droite répond à l’équation $x_1 + x_2 - 0.5 = 0$ et réalise un *OU* logique. La seconde répond à l’équation $x_1 + x_2 - 0.5 = 0$ et réalise un *ET* logique. Elles sont faites par la première couche du réseau. La seconde couche combine les deux décisions linéaires en une décision non linéaire.

10.2.5 Le codage des exemples d’apprentissage

Les modèles connexionnistes dont nous parlons utilisent pour leur apprentissage des données couplées à une classe d’appartenance, laquelle est représentée par un autre vecteur de *sorties désirées* noté \mathbf{u} (conformément aux notations du chapitre 2; voir aussi la page de notations, page xxvii); ce vecteur est de dimension égale aux nombre C de classes. Un exemple d’apprentissage \mathbf{z} est donc composé d’un vecteur \mathbf{u} de sortie désirée associé à un vecteur d’entrée \mathbf{x} : $\mathbf{z} = (\mathbf{x}, \mathbf{u})$. Par conséquent, chaque neurone formel de sortie correspond à une classe et une seule⁵.

L’apprentissage d’une règle de classification se fera en général en attribuant une classe à chaque coordonnée du vecteur des sorties: la classe ω_1 sera codée $\mathbf{u}^T = (1, 0, 0, \dots, 0)$, la classe ω_C : $\mathbf{u}^T = (0, 0, \dots, 1)$.

Un réseau connexionniste multicouche peut en réalité apprendre des associations plus complexes: la sortie étant un vecteur de \mathbb{R}^C , il peut approximer toute fonction de \mathbb{R}^d dans \mathbb{R}^C .

10.3 L’algorithme d’apprentissage

La caractéristique la plus intéressante d’un réseau de neurones artificiels est sa capacité d’apprendre, c’est-à-dire de modifier les poids de ses connexions en fonction des données d’apprentissage, de telle sorte qu’après un certain temps d’entraînement il ait acquis une faculté de généralisation.

Pour procéder graduellement, nous allons d’abord redécrire l’algorithme du perceptron, en le considérant comme un réseau connexionniste. Nous montrerons ensuite que cet algorithme d’apprentissage peut se voir comme un problème d’optimisation qui se résout par une méthode de gradient. Ce qui permettra de généraliser d’abord à un perceptron travaillant avec plus de deux classes, puis au réseau connexionniste multicouche.

5. Il existe d’autres codages dans lesquels une classe est associée à un sous-ensemble de neurones de sortie. Ceci permet en particulier d’utiliser la technique des codes correcteurs d’erreur (voir [DB95]).

10.3.1 Retour sur le perceptron

10.3.1.1 Le perceptron pour deux classes

Fonction seuil en sortie

Le *perceptron* a été déjà été étudié au chapitre 9 dans le cadre des séparateurs linéaires. On va le voir ici comme un réseau connexionniste à couches, comportant une couche de neurones d'entrées, un neurone de sortie unique et pas de couche cachée. Les connexions sont donc faites directement entre la couche d'entrée et le neurone de sortie, ce qui se traduit dans le cas de deux classes par une décision par seuil sur une combinaison linéaire des valeurs d'entrée.

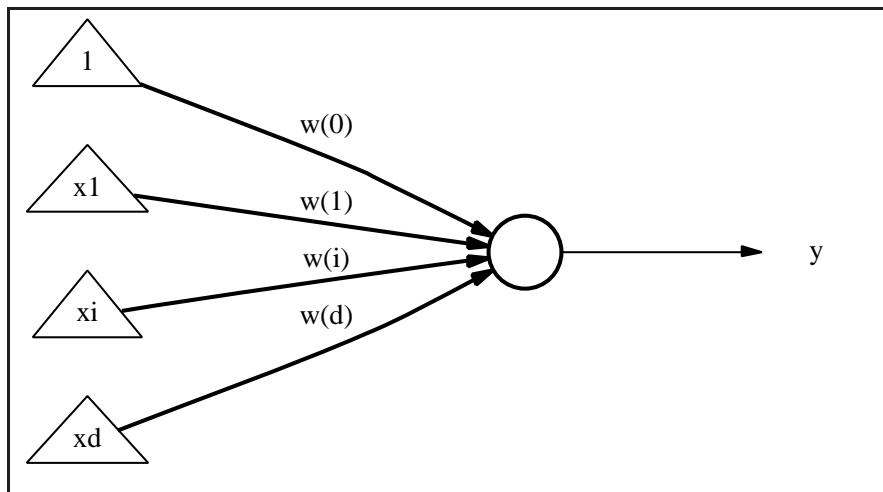


FIG. 10.12 – *Le perceptron est le réseau connexionniste le plus simple. Il effectue le calcul : $y = f(\sigma) = w(0) + \sum_{i=1}^d w(i)x_i$*

La figure 10.12 montre comment le perceptron peut être représenté comme un réseau connexionniste sans couche cachée, avec un seul neurone formel de sortie.

f est ici la fonction seuil, calculée à partir de σ de la manière suivante :

$$\sigma \geq 0 \Leftrightarrow y = f(\sigma) = 1$$

$$\sigma \leq 0 \Leftrightarrow y = f(\sigma) = 0$$

L'apprentissage dans le perceptron se fait par la règle de modification des poids qui a été donnée au chapitre 9. Dans le cas de deux classes, il n'y a qu'une seule sortie, et la décision d'appartenance à une des deux classes est prise en comparant la valeur de sortie à un seuil. L'algorithme d'apprentissage se contente de modifier le vecteur des poids en lui ajoutant ou lui enlevant un vecteur proportionnel à l'entrée x , dans le cas où celle-ci conduit à une valeur du mauvais côté du seuil ; il ne fait rien sinon.

Au chapitre 9, nous avons formalisé cet apprentissage, pour deux classes ω_1 et ω_2 par l'algorithme 9.2.

Récrivons-le un peu différemment pour nous placer dans les notations de ce chapitre. Le vecteur a du chapitre 9 correspond directement à l'ensemble des poids des connexions. Nous notons ici $w(i)$ le poids de la connexion menant de l'entrée x_i au neurone de sortie.

\mathbf{u} , $\boldsymbol{\sigma}$ et \mathbf{y} sont pour le moment des vecteurs ayant une seule composante. Le perceptron effectue donc le calcul fondé sur l'équation 10.2 :

$$\mathbf{y} = f(\boldsymbol{\sigma}) = f \left(w(0) + \sum_{i=1}^d w(i)x_i \right)$$

En phase d'apprentissage, la modification apportée à $w(i)$ au cours de l'apprentissage pour le mener de sa valeur à l'instant t à celle à l'instant $t+1$ par une entrée \mathbf{x} se note Δ_i ; elle peut maintenant s'écrire de la manière suivante :

$$\Delta_i = \alpha x_i (\mathbf{u} - \mathbf{y}) \quad (10.3)$$

En effet, quand \mathbf{x} est bien classé, le terme $(\mathbf{u} - \mathbf{y})$, qui est un scalaire (les vecteurs \mathbf{u} et \mathbf{y} sont de dimension 1), vaut 0. Quand \mathbf{x} est mal classé, ce terme vaut +1 ou -1 selon que \mathbf{x} est un exemple ou un contre-exemple. L'algorithme d'apprentissage devient donc :

Algorithme 10.1 Reformulation de l'apprentissage du perceptron.

Prendre $a_{(0)}$ quelconque et α positif quelconque

$t = 0$

tant que $t \leq t_{max}$ **faire**

tirer au hasard une donnée d'apprentissage \mathbf{x} parmi les m

pour $i = 1, d$ **faire**

$\Delta_i \leftarrow \alpha x_i (\mathbf{u} - \mathbf{y})$

$w_i \leftarrow w_i + \Delta_i$

fin pour

$t \leftarrow t + 1$

fin tant que

Fonction identité en sortie

Supposons que nous enlevions la fonction seuil à la sortie et que l'on ait désormais :

$$\mathbf{y} \equiv \boldsymbol{\sigma}$$

Rien n'empêche d'appliquer la même modification de poids que précédemment, c'est-à-dire :

$$\Delta_i = \alpha x_i (\mathbf{u} - \boldsymbol{\sigma})$$

On se trouve alors dans un apprentissage du type « punition-récompense » : les poids sont modifiés proportionnellement à l'entrée et à la conformité de la sortie avec la sortie désirée. Pour être plus précis, si nous définissons par

$$E = \frac{1}{2}(\mathbf{u}^2 - \mathbf{y}^2) \quad (10.4)$$

la fonction d'erreur entre la sortie $\mathbf{y} = \sum_{i=0}^d w_i x_i$ et la sortie désirée \mathbf{u} , on peut constater que :

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial \mathbf{y}} \cdot \frac{\partial \mathbf{y}}{\partial w_i} = -(\mathbf{u} - \boldsymbol{\sigma}) x_i$$

Δ_i peut donc être interprétée comme une modification de w_i par la technique du gradient pour minimiser E .

Voyons maintenant comment on peut, en gardant le même principe d'apprentissage, généraliser le perceptron pour lui donner des capacités beaucoup plus fortes, selon deux directions : en augmentant la taille de sa couche de sortie, puis en introduisant des couches cachées.

10.3.1.2 Le perceptron pour plus de deux classes.

La transformation précédente permet maintenant d'appliquer directement le même calcul à l'apprentissage d'une règle de classification pour un nombre quelconque C de classes. Il suffit, comme on l'a vu, de construire pour chaque donnée d'apprentissage un vecteur de sortie désirée \mathbf{u} de dimension C valant 1 sur la coordonnée correspondant à la classe et 0 partout ailleurs (figure 10.13).

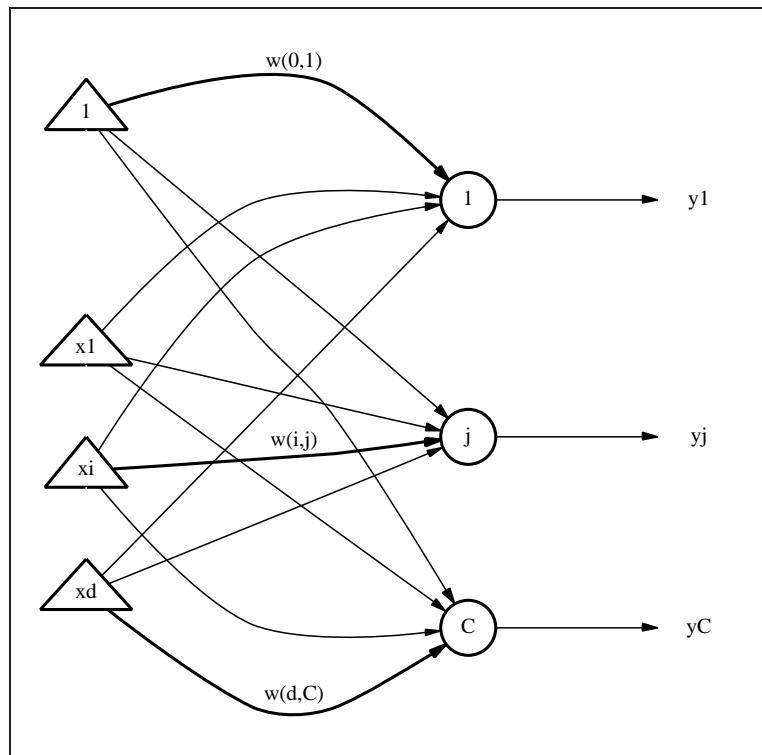


FIG. 10.13 – Le perceptron pour C classes.

En notant $w(i, j)$ le poids de la connexion menant de l'entrée d'indice i à la sortie d'indice j , on peut calculer pour une entrée \mathbf{x} donnée chaque sortie par :

$$y_j = \sigma_j = w(0, j) + \sum_{i=1}^{i=d} w(i, j)x_i$$

Idéalement, un perceptron ayant parfaitement appris (au moins) les données d'apprentissage devrait fournir pour chacune d'elles une sortie $\mathbf{y} = \boldsymbol{\sigma}$ égale à la sortie désirée \mathbf{u} .

On mesure alors l'*erreur de classification pour l'entrée x* comme la distance euclidienne⁶ $D(\mathbf{u}, \mathbf{y})$ entre la sortie désirée \mathbf{u} et la sortie calculée \mathbf{y} :

$$D(\mathbf{u}, \mathbf{y}) = D(\mathbf{u}, \boldsymbol{\sigma}) = \frac{1}{2} \sum_{j=1}^{j=C} (u_j - \sigma_j)^2 \quad (10.5)$$

6. On note dans ce chapitre la distance euclidienne D et non pas Δ , pour éviter la confusion avec la notation traditionnelle de la « règle delta » de l'apprentissage des réseaux connexionnistes.

On peut appliquer pour l'apprentissage une généralisation de la technique du paragraphe précédent : la *règle delta*. Elle consiste à modifier le poids $w(i, j)$ d'une quantité :

$$\Delta_{ij} = \alpha x_i (u_j - y_j) \quad (10.6)$$

où α est une valeur positive comprise entre 0 et 1.

Ceci revient encore à appliquer la technique de l'*optimisation par gradient* (voir l'annexe 18.2), qui consiste ici à constater que la contribution du poids $w(i, j)$ à l'erreur $D(\mathbf{u}, \mathbf{y})$ peut s'écrire :

$$\frac{\partial}{\partial w(i, j)} D(\mathbf{u}, \sigma)$$

Puisque l'on a :

$$\sigma_j = w(0, j) + \sum_{i=1}^{i=d} w(i, j) x_i$$

et :

$$D(\mathbf{u}, \mathbf{y}) = D(\mathbf{u}, \sigma) = \frac{1}{2} \sum_{j=1}^{j=C} (u_j - \sigma_j)^2$$

le calcul se fait donc ainsi :

$$\begin{aligned} \frac{\partial}{\partial w(i, j)} D(\mathbf{u}, \sigma) &= \frac{\partial}{\partial \sigma_j} D(\mathbf{u}, \sigma) \frac{\partial \sigma_j}{\partial w(i, j)} \\ &= \frac{1}{2} \left(\frac{\partial}{\partial \sigma_j} (u_j - \sigma_j)^2 \right) \frac{\partial \sigma_j}{\partial w(i, j)} \\ &= (\sigma_j - u_j) x_i \\ &= (y_j - u_j) x_i \end{aligned}$$

Selon la technique du gradient, $w(i, j)$ doit être modifié d'une valeur Δ_{ij} proportionnellement et en sens inverse à la contribution du poids $w(i, j)$ à l'erreur $D(\mathbf{u}, \mathbf{y})$. D'où la formule 10.6 donnée ci-dessus pour la règle delta.

La règle delta est donc à la fois une généralisation de la règle d'apprentissage du perceptron pour le cas à deux classes et une technique de minimisation par gradient de l'erreur quadratique moyenne.

10.3.1.3 Plus de deux classes et une sigmoïde en sortie

Dans ce paragraphe, nous continuons à progresser en supposant que la sortie σ_j est transformée en $y_j = f(\sigma_j)$, où f est la fonction sigmoïde de paramètre $\lambda = 1$ (voir la figure 10.4) :

$$y_j = f(\sigma_j) = \frac{1}{1 + e^{-\sigma_j}}$$

Le calcul devient :

$$\frac{\partial}{\partial w(i, j)} \left(\frac{1}{2} (u_j - y_j)^2 \right) = \frac{1}{2} \left(\frac{\partial}{\partial y_j} (u_j - y_j)^2 \right) \frac{\partial y_j}{\partial \sigma_j} \frac{\partial \sigma_j}{\partial w(i, j)} = (y_j - u_j) y_j (1 - y_j) x_i$$

D'où :

$$\Delta_{ij} = \alpha (u_j - y_j) y_j (1 - y_j) x_i \quad (10.7)$$

En effet, comme indiqué dans l'annexe 18.3, la fonction f répond à l'équation différentielle :

$$f' = f(1 - f)$$

10.3.2 L'apprentissage par rétropropagation du gradient de l'erreur

C'est seulement en 1986 que la généralisation de la règle delta aux réseaux à couches cachées a été formulé. Cette généralisation, *la règle de la rétropropagation du gradient de l'erreur*, consiste à propager l'erreur obtenue à une unité de sortie d'un *réseau à couches* comportant une ou plusieurs couches cachées à travers le réseau par descente du gradient dans le sens inverse de la propagation des activations. La figure 10.14 montre une illustration du principe.

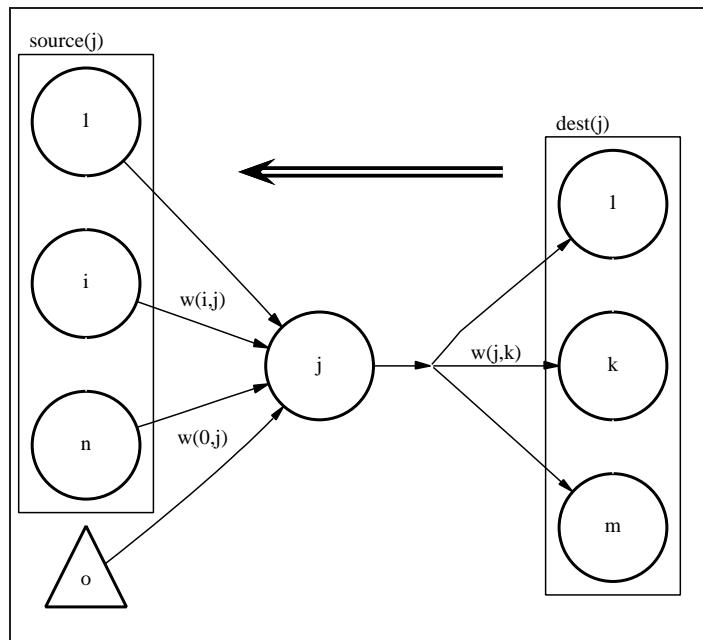


FIG. 10.14 – Schéma du modèle de la rétropropagation de l'erreur. La modification à apporter aux poids entre la couche *source(j)* et le neurone formel *j* ne peut être calculée que si on connaît déjà la modification qu'il faut apporter aux poids *w(j, k)* entre *j* et les éléments de *dest(j)*.

Rappelons qu'un réseau à couches est composé d'un ensemble de neurones formels groupés en sous-ensembles distincts (les couches) de telle sorte qu'il n'y ait aucune connexion entre deux neurones d'une même couche⁷.

À la fin de l'apprentissage, lorsque le réseau a appris à modéliser son environnement, le comportement souhaité du réseau est le suivant : on présente un vecteur d'entrée au réseau, celui-ci propage vers la sortie les valeurs d'activation correspondantes (en utilisant une règle de propagation), afin de générer, par l'intermédiaire des neurones de sortie, un vecteur de sortie. Celui-ci devrait correspondre à la sortie désirée, telle qu'apprise lors de la phase d'apprentissage.

La généralisation de la règle delta aux réseaux multicouches utilise une méthode de *descente du gradient*, permettant de calculer la modification des poids des connexions entre les couches cachées (pour plus de détails, voir l'annexe 18.3 ou la référence [RHW86]). Afin de pouvoir calculer le gradient de l'erreur par rapport aux poids du réseau, la fonction de sortie d'un neurone doit être différentiable et non linéaire (sinon, on pourrait réduire le réseau à un perceptron). La fonction la plus souvent utilisée est, comme on l'a déjà dit, la *sigmoïde* :

7. Une amélioration de la règle de rétropropagation permet l'introduction de cycles entre les couches, pour obtenir des réseaux récurrents.

$$y_j = f(\sigma_j) = \frac{1}{1 + e^{-\sigma_j}} \quad (10.8)$$

La règle delta généralisée dicte alors le changement de poids entre le neurone i et le neurone j de la façon suivante :

$$\Delta w(i, j) = \alpha \delta_j y_i \quad (10.9)$$

c'est-à-dire de façon proportionnelle à une mesure d'erreur δ_j caractéristique du neurone j et à la valeur d'entrée notée ici⁸ y_i . Pour les connexions aboutissant aux neurones de sortie, cette mesure d'erreur est évidemment calculée ainsi :

$$\delta_j = (u_j - y_j) y_j (1 - y_j) \quad (10.10)$$

Le calcul de l'erreur aux unités cachées se fait ensuite récursivement par la descente du gradient. Soit $dest(j)$ l'ensemble des neurones auxquels j se connecte :

$$\delta_j = y_j (1 - y_j) \sum_{k \in dest(j)} \delta_k w(j, k) \quad (10.11)$$

Le calcul est détaillé dans l'annexe 18.3.

Lorsque l'on applique la règle delta généralisée sur le réseau de façon itérative pour un ensemble de vecteurs d'entrées (correspondant à l'environnement), le réseau tentera de minimiser l'erreur obtenue à la sortie, et donc de modéliser le mieux possible la fonction désirée entre les entrées et les sorties.

10.3.3 L'organisation des calculs

Les calculs s'organisent de la façon donnée dans l'algorithme 10.2. Le point à remarquer est que l'actualisation des poids ne se fait qu'une fois la rétropropagation terminée : il ne faut en effet pas changer trop tôt la valeur d'un poids puisque celle-ci intervient dans le calcul concernant la couche suivante.

10.3.4 Retour sur l'exemple

Reprendons l'exemple du paragraphe 10.7 en supposant que la sortie désirée au vecteur d'entrée $x^T = (1, 1)$ vaille $u = 0$. Après modification des poids sur cet exemple, son nouveau passage dans le réseau doit conduire à une sortie inférieure à la valeur précédente, qui était de 0.63.

Pour le neurone formel de sortie, on a :

$$\Delta w(i, j) = \alpha \delta_j y_i$$

avec :

$$\delta_j = (u_j - y_j) y_j (1 - y_j)$$

On prend d'abord : $i = 3$ et $j = 5$, ce qui mène à :

$$\delta_5 = (0. - 0.63) \times 0.63 \times (1. - 0.63) = -0.147$$

8. C'est en effet la sortie du neurone i .

Algorithme 10.2 Apprentissage du perceptron multicouche.

```

tant que l'apprentissage n'a pas convergé faire
    tirer au hasard un point d'apprentissage
    pour chaque couche, en partant de celle du haut faire
        pour chaque neurone formel de cette couche faire
            calculer  $\delta_j$ 
            pour chaque connexion  $w(i, j)$  menant au neurone formel  $j$  faire
                calculer  $\Delta w(i, j) = \alpha \delta_j y_i$ 
            fin pour
        fin pour
    fin pour
    pour chaque connexion  $w(i, j)$  faire
         $w(i, j) \leftarrow w(i, j) + \Delta w(i, j)$ 
    fin pour
fin tant que

```

d'où :

$$\Delta w(3, 5) = -0.147 \times 0.65 \simeq -0.1$$

en fixant la valeur α à 1. De même, pour $i = 4$, on obtient :

$$\Delta w(4, 5) = 0.48 \times -0.147 \simeq -0.07$$

$$\Delta w(0, 5) = -0.147 \times 1. = -0.147$$

Pour le neurone formel caché noté 4, on a d'abord, puisque $dest(4) = \{5\}$:

$$\delta_4 = y_4 \times (1 - y_4) \times \delta_5 \times w(4, 5) = 0.48 \times (1 - 0.48) \times -0.147 \times -0.4 = \simeq 0.015$$

D'où :

$$\Delta w(1, 4) = 0.015 \times 1. = 0.015$$

$$\Delta w(2, 4) = 0.015 \times 1. = 0.015$$

$$\Delta w(0, 4) = 0.015 \times 1. = 0.015$$

De même, puisque $dest(3) = \{5\}$:

$$\delta_3 = y_3 \times (1 - y_3) \times \delta_5 \times w(3, 5) = 0.65 \times (1 - 0.65) \times -0.147 \times 0.5 = \simeq -0.017$$

D'où :

$$\Delta w(1, 3) = 0.016 \times 1. = -0.017$$

$$\Delta w(2, 3) = 0.016 \times 1. = -0.017$$

$$\Delta w(0, 3) = 0.016 \times 1. = -0.017$$

Après modification, les poids deviennent donc :

$$w(0, 5) + \Delta w(0, 5) = 0.4 - 0.147 \simeq 0.25$$

$$w(3, 5) + \Delta w(3, 5) = 0.5 - 0.1 = 0.4$$

$$w(4, 5) + \Delta w(4, 5) = -0.4 - 0.07 = -0.47$$

$$w(0, 3) + \Delta w(0, 3) = 0.2 - 0.017 = 0.183$$

$$w(1, 3) + \Delta w(1, 3) = 0.1 - 0.017 = 0.083$$

$$w(2, 3) + \Delta w(2, 3) = 0.3 - 0.017 = 0.283$$

$$w(0, 4) + \Delta w(0, 4) = -0.3 + 0.015 = -0.285$$

$$w(1, 4) + \Delta w(1, 4) = -0.2 + 0.015 = -0.185$$

$$w(2, 4) + \Delta w(2, 4) = 0.4 + 0.015 = 0.415$$

Dans le réseau modifié, le calcul sur le vecteur d'entrée devient par conséquent :

Neurone formel j		σ_j	y_j
3	$0.183 + 0.083 \times 1 + 0.283 \times 1 = 0.55$	$\frac{1}{1+e^{0.65}} \simeq \mathbf{0.63}$	
4	$-0.285 + -0.185 \times 1 + 0.415 \times 1 = -0.055$	$\frac{1}{1+e^{-0.055}} \simeq \mathbf{0.51}$	
5	$0.25 + 0.4 \times \mathbf{0.63} - 0.47 \times \mathbf{0.51} = 0.26$	$\frac{1}{1+e^{0.205}} \simeq 0.56$	

Si on compare la valeur de sortie à celle du tableau de la section 10.2.2 on constate qu'elle est passée de 0.63 avant apprentissage à 0.56 après : elle s'est rapprochée de la valeur désirée 0.

10.3.5 Une variante

Il est possible de transformer un réseau connexionniste multicouche en un système de décision bayésien (voir les chapitres 2 et 14) en changeant la distance entre la sortie calculée et la sortie désirée, sans modifier la règle de rétropropagation du gradient de l'erreur.

On désire ici une valeur du neurone de sortie y_j qui vaille la probabilité que le vecteur d'entrée appartienne à la classe j . Soit $X = (X_1, \dots, X_C)$ une variable aléatoire multidimensionnelle qui représente, sous une hypothèse multinomiale, la distribution des sorties désirées. La probabilité de X s'exprime en fonction des probabilités *a priori* y_j d'appartenir à la classe j :

$$P(X_1 = u_1, \dots, X_C = u_C) = \prod_{j=1}^C y_j^{u_j} (1 - y_j)^{\sum_{i=1}^C u_i - u_j}$$

Au maximum de vraisemblance, chercher les paramètres qui maximisent cette quantité est équivalent à minimiser la fonction d'erreur *entropie croisée* (voir le chapitre 11) :

$$E = \sum_{j=1}^C -u_j \cdot \text{Log}(y_j) - (1 - u_j) \cdot \text{Log}(1 - y_j) \quad (10.12)$$

Il faut calculer :

$$\frac{\partial E}{\partial w(i, j)} = \frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial \sigma_j} \cdot \frac{\partial \sigma_j}{\partial w(i, j)}$$

On a :

$$\begin{aligned}\frac{\partial E}{\partial y_j} &= -\frac{u_j}{y_j} + \frac{1-u_j}{1-y_j} = \frac{y_j - u_j}{y_j(1-y_j)} \\ \frac{\partial y_j}{\partial \sigma_j} &= y_j(1-y_j) \\ \frac{\partial \sigma_j}{\partial w(i,j)} &= y_i\end{aligned}$$

Finalement :

$$\frac{\partial E}{\partial w(i,j)} = (y_j - u_j) \cdot y_i \quad (10.13)$$

Il ne reste plus qu'à appliquer la règle de rétropagation du gradient de l'erreur avec cette valeur, au lieu de la règle delta de l'équation 10.7, qui correspond à la distance euclidienne entre la sortie calculée et la sortie désirée.

10.3.6 Quand arrêter l'apprentissage ?

Il est difficile de trouver un critère général pour arrêter cet algorithme. Le problème est que le risque empirique tend à diminuer lentement et à ne jamais se stabiliser complètement, ce qui mène à un surapprentissage. La meilleure manière d'éviter ce phénomène est d'utiliser un ensemble de validation (voir chapitre 3, paragraphe 3.4.5.1).

10.3.7 Le problème des minima locaux

Comme tous les algorithmes d'optimisation basés sur la descente du gradient, l'algorithme de la rétropagation est susceptible de s'arrêter aux minima locaux. Par conséquent, la solution trouvée sera fortement reliée au choix des poids initiaux du réseau. Si les poids sont choisis près d'un minimum local sous-optimal, l'algorithme ne pourra pas trouver la solution désirée. Afin de contourner ce problème, on peut utiliser plusieurs techniques :

- Relancer l'apprentissage plusieurs fois en utilisant des poids initiaux différents, ce qui entraîne un temps de calcul plus élevé.
- Introduire du bruit dans la recherche pour pouvoir sortir des minima locaux.
- Utiliser les techniques avancées de descente de gradient : second ordre, gradient conjugué, etc. (voir [Bis95, Hay99]).

10.4 Quelques résultats théoriques sur les réseaux connexionnistes

Plusieurs résultats théoriques sur l'apprentissage des réseaux connexionnistes ont été obtenus, particulièrement en ce qui a trait à leur pouvoir d'expression, leur complexité, ainsi que leur capacité de généralisation. Nous donnons dans cette section quelques-uns de ces résultats.

10.4.1 Pouvoir d'expression

Le *pouvoir d'expression* d'un réseau de neurones connexionniste, comme de toute règle de classification, est une mesure du nombre de fonctions différentes que celui-ci peut approximer.

Il est en effet intéressant de connaître *a priori* les familles de fonctions auxquelles vont appartenir les surfaces de décision. Plusieurs résultats montrent par exemple qu'un réseau de neurones artificiels multicouche peut approximer avec une précision arbitraire n'importe quelle transformation continue d'un espace à dimension finie vers un autre espace à dimension finie, s'il possède suffisamment de neurones formels cachés (voir [Hay99]). En ce sens, on dit qu'il est un *approximateur universel*. Certains résultats montrent même qu'à l'exception de cas extrêmes, une seule couche cachée est suffisante.

Il faut cependant noter que ces résultats ne fournissent aucun indice sur la méthode à utiliser pour trouver directement les poids correspondant à l'approximation d'une fonction donnée. On ne sait les calculer que par apprentissage. Ce résultat était connu avant la découverte de l'algorithme de rétropropagation du gradient de l'erreur, qui a alors permis de l'utiliser en pratique.

10.4.2 Complexité

Les réseaux connexionnistes ayant un si grand pouvoir d'expression, il devient intéressant de connaître les aspects de complexité reliés à ce modèle (voir [Orp92] pour une revue). Ainsi, il a été montré le résultat suivant :

Étant donné un réseau de neurones artificiels arbitraire R et une tâche arbitraire T devant être résolue par R , le problème consistant à décider si dans l'espace de tous les paramètres de R (ses poids, sa structure) il existe une solution qui résout adéquatement T , est *NP-difficile*.

Malgré cela, il est possible [Bau89] de trouver une solution (un ensemble de poids) pour T en temps polynomial si on peut utiliser des algorithmes d'apprentissage *constructifs*⁹. Il existe un certain nombre de ces algorithmes mais aucune preuve de convergence en temps polynomial n'existe actuellement pour ces algorithmes.

D'un point de vue pratique, certaines expériences empiriques (dont [Hin89]) montrent qu'on peut faire apprendre une tâche complexe à un réseau de neurones artificiels en utilisant l'algorithme de la rétropropagation de l'erreur en temps $\mathcal{O}(W^3)$ où W représente le nombre de poids du réseau. En effet, bien qu'il faille un temps exponentiel (sur le nombre de poids) pour obtenir la solution optimale, on peut souvent en pratique se contenter d'une solution sous-optimale satisfaisante obtenue en temps polynomial.

10.4.3 Réseaux connexionnistes et apprentissage *PAC*

Des liens profonds existent entre la capacité d'apprentissage d'un réseau connexioniste et la théorie de l'apprentissage. Le lecteur peut se reporter par exemple à C. Bishop ([Bis95]) pour un traitement de ce sujet.

9. ayant la possibilité d'ajouter des neurones et des connexions durant l'apprentissage.

10.5 Comment choisir l'architecture d'un réseau?

Un des problèmes majeurs des réseaux connexionnistes a trait à la difficulté de décider de leur architecture. Devant une tâche à résoudre par un réseau connexionniste, l'ingénieur doit prendre des décisions d'architecture non évidentes et pourtant très importantes : par exemple, il faut décider du nombre de neurones cachés, du nombre de couches cachées, et de leur interconnexion. Ceci se fait souvent de façon *ad hoc* ou en utilisant quelques règles heuristiques simples. Souvent on procède en essayant diverses architectures pour un problème donné et en calculant l'erreur de généralisation pour chacune sur un ensemble de validation. En effet, hormis une recherche exhaustive, aucune méthode n'est connue pour déterminer l'architecture optimale pour un problème donné. Or tous les résultats théoriques sur les réseaux connexionnistes (leur puissance de calcul ou leur faculté de généralisation) ne tiennent que si l'on utilise l'architecture idéale (ou tout au moins suffisante et nécessaire).

Une solution à ce problème consiste à utiliser des algorithmes *constructifs* qui commencent avec une architecture minimale et ajoutent des neurones et des connexions au fur et à mesure de l'apprentissage. D'autres solutions utilisent plutôt une technique inverse : à partir d'une architecture complète, ils éliminent certains neurones et/ou connexions qui semblent non essentiels.

Depuis peu, on commence à utiliser des méthodes d'optimisation pour chercher l'architecture idéale. Ainsi, plusieurs travaux proposent l'utilisation des algorithmes génétiques pour optimiser l'architecture des réseaux de neurones (voir le chapitre 8).

Un autre problème tient au choix des paramètres des divers algorithmes d'apprentissage. En effet, chaque règle d'apprentissage utilise généralement un certain nombre de paramètres pour guider l'apprentissage. Ainsi, la règle de la rétropropagation de l'erreur est basée notamment sur le taux d'apprentissage noté α dans ce chapitre. Ce taux varie d'une tâche à l'autre, et encore une fois, on utilise souvent des règles heuristiques simples pour déterminer sa valeur idéale. Dans la même veine que pour le choix des architectures, on utilise maintenant des méthodes comme les algorithmes génétiques pour choisir ces paramètres.

Notes historiques et sources bibliographiques

L'ouvrage de Dreyfus et al. ([DMS⁺02]) est recommandé : encyclopédique, écrit en français et récent. Les livres de R. Golden ([Gol96]), C. Bishop ([Bis95]) et B. Ripley ([Rip96])) fournissent des panoramas sur la théorie et de la pratique de ces outils. Le chapitre 6 du livre de R. Duda, P. Hart et R. Stork ([DHS01]) est une remarquable introduction théorique et pratique. On pourra aussi consulter les ouvrages suivants, intéressants à divers titres : [Sch92], [WK91], [HKP91].

Le travail fondateur le plus souvent cité est celui de W. Mc Culloch et W. Pitt ([MP43]) bien que l'apprentissage n'y soit pas réellement abordé. On ne peut que remarquer le travail de A. Turing, en 1948, sur l'organisation d'unités logiques élémentaires en un ensemble au comportement complexe ([Tur92]). Les travaux sur les réseaux de neurones formels, souvent inspirés par des motivations de modélisation biologique, n'ont pas cessé depuis cette époque jusqu'à nos jours. Du point de vue de l'apprentissage artificiel, le tournant se situe en 1986, où il s'est produit un phénomène courant en sciences : la découverte indépendante et quasi simultanée d'un résultat important, en l'occurrence, les formules de la rétropropagation de gradient de l'erreur. Le livre de P. Werbos ([Wer84]) et l'article de B. Widrow ([Wid94]) relatent l'historique de cette découverte et des réseaux connexionnistes en général.

Une véritable explosion de publications et d'applications très diverses a suivi et continue de nos jours, couvrant aussi bien les améliorations des techniques d'apprentissage que leur généralisation à des modèles plus complexes, ou leur application à des données séquentielles

ou bidimensionnelles: signaux et images. Les liens des réseaux connexionnistes avec la théorie bayésienne, les théories de l'apprentissage et l'analyse (au sens mathématique du terme) ont aussi été éclaircis. Les travaux de modélisation biologiques et situés dans le domaine des sciences cognitives sont également très nombreux.

Le texte de ce chapitre est en partie inspiré de l'introduction de la thèse (PhD) de Samy Bengio, université de Montréal, 1993.

Résumé

- Les réseaux connexionnistes sont des mécanismes de calcul permettant en particulier d'affecter une classe à un vecteur de données numériques.
- Par un processus d'apprentissage par optimisation, un réseau connexionniste peut adapter ses paramètres à un ensemble de données supervisées, dans le but d'en généraliser les caractéristiques.
- Les capacités de généralisation de ces systèmes sont bonnes. L'algorithme d'apprentissage classique, la *rétropropagation du gradient de l'erreur*, est éprouvé. Il a donné lieu à de nombreuses améliorations, en particulier en ce qui concerne sa rapidité et son arrêt avant la surgénéralisation.
- Les réseaux connexionnistes ont été étendus à des architectures permettant de réaliser l'apprentissage de règles de classification de séquences. Ils permettent aussi l'apprentissage de fonctions de régression.

Chapitre 11

Apprentissage par combinaison de décisions

L'une des grandes familles d'approches pour la résolution de problèmes comme pour l'apprentissage est la technique consistant à « diviser pour régner » (divide and conquer). Elle se résume à identifier des sous-problèmes, à leur trouver une solution, puis à combiner ces solutions pour résoudre le problème général.

C'est sur ce principe que sont fondés les algorithmes d'apprentissage par arbres de décision. Ils apprennent à identifier les sous-espaces de l'espace d'entrée pour lesquels la solution est identique. Lorsqu'un nouveau cas est soumis au système, celui-ci identifie le sous-espace correspondant et retourne la réponse associée.

D'autres familles d'algorithmes distribuent la tâche entre plusieurs experts et combinent les solutions partielles pour obtenir la solution générale. L'apprentissage consiste alors à déterminer les solutions partielles et à trouver une manière efficace pour les combiner. Lorsque la combinaison des réponses des experts ne dépend pas de l'entrée, on parle de combinaison statique. Les méthodes de boosting, très étudiées actuellement, sont l'archétype de cette approche. Lorsque la combinaison dépend de l'entrée, on parle alors de structures dynamiques, dont les mélanges d'experts et les mélanges hiérarchiques d'experts sont les méthodes les plus représentatives.

LES MANUELS d'ornithologie et les « flores » (les livres pour l'identification des plantes à fleurs) ne sont en général pas organisés de la même manière. Pour les premiers, on trouve les oiseaux dans un ordre invariable, correspondant à l'énumération savante des ordres et des espèces. Un dessin de chaque oiseau est donné, accompagné d'une description imagée et de détails permettant de contraster l'espèce en question d'avec celles qui peuvent prêter à confusion. Par exemple, si on observe un grand oiseau blanc sur un plan d'eau, un rapide parcours des figures amène sans ambiguïté à la page des cygnes, où seulement trois espèces sont décrites¹. Il reste à se décider avec le texte et les détails des dessins, en lisant par exemple : « Au repos, le cygne tuberculé a l'habitude de tenir le cou recourbé », ou : « Le cygne chanteur a un bec jaune ; il est migrateur hivernal et niche dans la toundra ».

En ce qui concerne les flores, une organisation différente est souvent adoptée. La raison principale est qu'il y a beaucoup plus d'espèces de plantes à fleurs que d'oiseaux. À l'observation d'une fleur, il est impossible de parcourir au hasard des milliers d'illustrations en espérant trouver la bonne page avec une probabilité suffisante. C'est pourquoi le système de recherche est fondé sur un questionnaire structuré. Supposons avoir devant nous une certaine fleur et dans la main une édition de la flore Bonnier². Une séquence de reconnaissance (un peu raccourcie pour la lisibilité) sera par exemple la suivante, en supposant que la fleur « répond » positivement à chaque test :

- *Plante ayant des fleurs, avec des étamines ou un pistil, ou les deux à la fois ?*
- *Fleurs non réunies en capitule entouré d'une collerette de bractées ?*
- *Fleurs à deux enveloppes de couleur et de consistance différentes ?*
- *Corolle non papilionacée ?*
- *Pétales libres entre eux ?*
- *Fleur ayant plus de douze étamines ?*
- *Étamines réunies entre elles ?*
- *Plante herbacée ?*
- *Fleurs à l'aisselle des feuilles ?*
- *Calicule à trois bractées libres et stigmate obtus ?*
- *Une seule fleur à l'aisselle des feuilles ?*
- *Bractées du calicule étroites ; carpelle velue ?*

Décision : la plante est la *Malva rotundifolia L.* (“Mauve à feuille rondes”)

À chaque question, qui porte sur un attribut de la fleur, la réponse est donc positive ou négative. Si aucune erreur n'est commise dans ces réponses, l'identification est réalisée.

Évidemment, les premiers chapitres de la flore Bonnier sont consacrés à des notions sur l'anatomie des plantes. Un index, comportant des mots comme « carpelle », « bractée », est également fourni. La difficulté est pour le lecteur de prendre une décision sans erreur pour chaque question posée. Le problème est d'organiser l'ensemble des questions de manière aussi efficace que possible, c'est-à-dire d'éviter les questions inutiles et de veiller à ce que chaque plante, en moyenne, puisse être identifiée par le plus petit nombre possible de questions.

1. On parle ici de manuels concernant l'avifaune européenne.

2. *Nouvelle flore pour la détermination facile des plantes, sans mots techniques, représentant toutes les espèces vasculaires des environs de Paris dans un rayon de 100 Kilomètres, des départements de l'Eure, de l'Eure et Loire, etc... Ouvrage couronné par l'académie des sciences et par l'académie d'agriculture de France. Par G. Bonnier, membre de l'institut et professeur à la Sorbonne, et Georges de Layens, lauréat de l'académie des sciences. Quatorzième édition, augmentée (...); Librairie générale de L'enseignement, 1926.*

11.1 Les arbres de décision

11.1.1 Principe

La technique des *arbres de décision* est fondée sur l'idée simple de réaliser la classification d'un objet par une suite de tests sur les attributs qui le décrivent. Ces tests sont organisés de telle façon que la réponse à l'un d'eux indique à quel prochain test auquel on doit soumettre cet objet.

Ce type de classification est, comme on l'a vu, couramment employé en sciences naturelles. Dans ce cas, l'espace de représentation est défini par l'observation des caractéristiques anatomiques utiles de la plante (*étamines, corolle, calicule, bractées, etc.*) ainsi que de leur existence conjointe, position relative, nombre, topologie, etc. Il faut connaître la signification et la mesure d'une bonne centaine de tels termes (c'est la taille de l'espace de représentation, le nombre d'attributs) pour classer toute plante répertoriée, dont le nombre d'espèces possibles est ici de 1500 à 2000 (c'est le nombre de classes).

Le principe de cette règle de décision est d'organiser l'ensemble des tests possibles comme un arbre³. Une feuille de cet arbre désigne une des C classes (mais à chaque classe peut correspondre plusieurs feuilles) et à chaque nœud est associé un test (un *sélecteur*) portant sur un ou plusieurs attributs, éléments de l'espace de représentation ; la réponse à ce test désignera le fils du nœud vers lequel on doit aller. La classification s'effectue donc en partant de la racine pour poursuivre récursivement le processus jusqu'à ce qu'on rencontre une feuille. Une telle structure est appelée *arbre de décision*. La question qui nous intéresse particulièrement est de réaliser l'apprentissage de telles structures de décision à partir d'exemples. Prenons une illustration dans un tout autre univers pour approcher ce problème.

Un exemple

Supposons que j'aie à prendre la décision suivante: *vais-je sortir le chien ou non?* Pour cela, j'observe les attributs suivants :

- *Quel temps fait-il?* C'est un attribut nominal pouvant prendre les valeurs *pluvieux, ensoleillé* ou *couvert*.
- *Quelle est la température extérieure?* Cet attribut est numérique.
- *Est-ce que le voisin est parti en week-end avec son chat?* Cet attribut est binaire.

Mon expérience m'a prouvé que la présence du chat du voisin rend la promenade assez pénible ; mais je sais que cet animal déteste l'humidité. D'autre part, le retour d'un chien mouillé n'est pas très plaisant pour mon tapis. Pour finir, ajoutons que je suis plutôt frileux. Moyennant quoi, je peux par exemple organiser ma décision selon la hiérarchie de la figure 11.1.

Cet *arbre de décision* se lit ainsi : j'observe d'abord le ciel. Si je remarque que le temps est couvert je dois ensuite regarder le thermomètre pour me décider. Si le temps est ensoleillé, je dois alors m'intéresser à la présence de mon voisin. S'il pleut, ma décision est toute prise.

Quelques avantages

Si l'on connaît un arbre de décision associé à un problème de classification, on voit immédiatement les avantages de ce type de règle de classification :

- Le nombre moyen de tests à effectuer sur une forme peut être extrêmement réduit (si les d attributs sont tous binaires, ce nombre est limité par d).
- La structure de décision est globale : on n'a pas de problème pour traiter C classes.
- Le test de tous les attributs de chaque objet à chaque nœud n'est pas nécessaire ; dans la plupart des cas pratiques, on se limite même à un seul test.

3. La botanique n'a plus rien à voir ici.

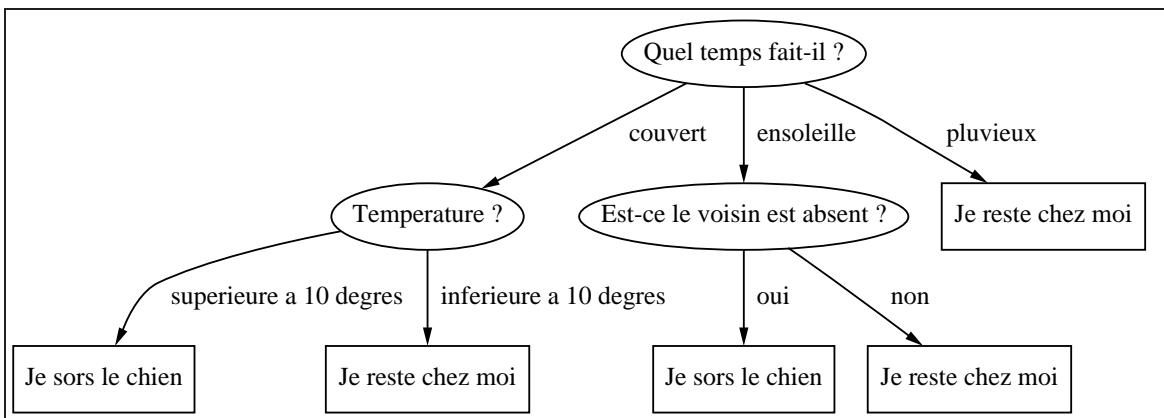


FIG. 11.1 – Un arbre de décision pour la promenade du chien.

Évidemment, ces avantages ne valent que s'il est possible de construire un arbre de décision à partir d'un ensemble d'apprentissage en remplissant au mieux deux conditions : celle de la proximité du *risque empirique* et du *risque réel* et celle de la *simplicité* de l'arbre obtenu, comme indiqué au chapitre 3.

11.1.2 La construction récursive d'un arbre de décision

11.1.2.1 Introduction

Dans l'exemple botanique proposé dans l'introduction, l'expertise joue un rôle très important : l'arbre de décision est construit à partir de connaissances sur la correspondance complexe entre les caractéristiques observables d'une plante et la définition de son espèce (fondée elle-même sur la possibilité de reproduction, la distribution géographique, etc.). La structure de cet arbre de décision est donc le résultat de l'expérience des botanistes. Mais pour réaliser la construction automatique d'un arbre de décision, il faut s'appuyer seulement sur un ensemble d'apprentissage et non pas sur une expertise. Comment sous cette hypothèse *apprendre* un arbre de décision performant en généralisation ?

Notons d'abord qu'il est hors de question d'explorer exhaustivement l'ensemble des arbres possibles pour déterminer le plus performant au sens d'un critère inductif comme l'*ERM* (chapitre 2) ou le principe de compression maximale (chapitre 17). En effet, le nombre d'arbres possibles est gigantesque, croissant exponentiellement avec le nombre d d'attributs et le nombre moyen a de valeurs possibles par attribut. Un calcul simple montre en effet que ce nombre est de :

$$\sum_{i=0}^{d-1} (d-i)^{a^i}$$

ce qui pour seulement quatre attributs à trois valeurs chacun donne déjà cinq cent vingt-six arbres possibles. Il faut donc un moyen « intelligent » d'explorer l'espace des hypothèses.

L'apprentissage des arbres de décision procède par une exploration du général au particulier en commençant par un arbre à un noeud racine correspondant à une partition simple de l'espace \mathcal{X} des exemples, puis en raffinant progressivement cette partition par ajout successif de nœuds dans l'arbre, ce qui revient à subdiviser itérativement les partitions de l'espace des exemples.

L'approche, appelée *induction descendante d'arbres de décision* (*top-down induction of decision tree*), procède de manière descendante, en partant de l'échantillon des données d'appren-

tissage toutes classes confondues. Tant que l'échantillon courant de données n'est pas « pur » (tous les exemples de la même classe) ou qu'il reste au moins un attribut à tester, un attribut est sélectionné, selon un critère décrit plus bas, pour servir de test premettant de subdiviser l'échantillon d'apprentissage courant en sous-échantillons distincts. À l'arrêt, on obtient donc un arbre de tests (nœuds) dont les feuilles correspondent à des échantillons d'exemples aussi « purs » que possible, c'est-à-dire idéalement appartenant à la même classe. Ce n'est pas en général possible, mais on garde l'idée de ramifier l'arbre autant qu'il le faudra pour arriver à une configuration où chaque feuille représente des données appartenant toutes à la même classe. Cette technique, basée sur le principe *ERM*, produit un arbre dont chaque feuille ne couvre plus qu'un faible nombre de données pures. Parce qu'il est trop dépendant des données d'apprentissage, on sait qu'il donnera vraisemblablement une mauvaise généralisation. C'est pourquoi on essaie de contrebalancer ce « surapprentissage » par un mécanisme limitant la complexité de l'arbre (donc du modèle) appris. On retrouve là le problème de la sélection de modèles (voir chapitres 2 et 3).

Si l'on a assez de données d'apprentissage, la façon la plus efficace est de procéder en deux passes : d'abord utiliser une partie \mathcal{A} de l'ensemble d'apprentissage pour construire un arbre T_{max} dont toutes les feuilles sont aussi pures que possible ; ensuite élaguer (simplifier) cet arbre avec une autre partie \mathcal{V} des données (un ensemble de *validation* comme défini au chapitre 3). Le reste des données, sous forme d'ensemble de test \mathcal{T} , sert enfin à évaluer le risque réel de l'arbre construit. Si les données sont peu nombreuses, une technique un peu plus complexe de validation croisée est nécessaire.

Au cours de la construction de T_{max} , le test mis en place à chaque nœud est basé sur le seul examen de la meilleure façon de séparer en classes le sous-ensemble considéré des points d'apprentissage qu'il régit. Le paragraphe suivant présente comment fabriquer de tels critères. On montrera ensuite comment élaguer T_{max} .

Pour simplifier l'exposé, nous commençons par le cas d'attributs binaires, mais tout ce qui suit est immédiatement généralisable au cas d'attributs multivalués.

11.1.2.2 Le cas des attributs binaires

Position du problème

On dispose d'un ensemble d'apprentissage \mathcal{S} de m exemples dont l'un est noté (\mathbf{x}, ω) .⁴ Cet exemple est décrit par d attributs $\{x_i, i = 1, d\}$ et par une classe $\omega \in \mathcal{C} = \{\omega_1, \dots, \omega_C\}$. On cherche d'abord, en appliquant le principe *ERM*, à construire un arbre de classification dont l'erreur apparente est nulle. On suppose pour l'instant que les attributs sont à valeur binaire, avant de considérer plus loin le cas où ils sont nominaux ou continus⁵.

L'algorithme de construction, décrit informellement ci-dessus, s'écrit récursivement :

Par conséquent, quand l'arbre est partiellement construit, à chaque nœud correspond un sous-ensemble des exemples d'apprentissage : ceux qui satisfont tous les tests binaires menant à ce nœud. Si ce sous-ensemble n'est pas constitué de points appartenant tous à la même classe, la construction doit se poursuivre. Il faut alors choisir le meilleur attribut à tester.

L'appel de cette procédure récursive se fait sur l'ensemble d'apprentissage \mathcal{S} . Il est à noter que dans certains cas, le test d'arrêt ne peut pas être satisfait : il peut exister plusieurs exemples ayant les mêmes attributs et des classes différentes⁶. Dans ce cas, la classe est attribuée par un

4. On n'a pas besoin ici d'indiquer les exemples dans l'ensemble d'apprentissage.

5. Les attributs à domaine arborescent ou séquentiels ne sont pas traités simplement par les arbres de décision.

6. Soit parce qu'il n'y a pas assez d'attributs pour les décrire et les discriminer, soit parce qu'il y a des erreurs de description ou d'étiquetage des exemples.

Algorithme 11.1 Construction récursive d'un arbre de décision

Procédure : Construire-arbre(X)

si Tous les points de X appartiennent à la même classe **alors**

Créer une feuille portant le nom de cette classe

sinon

Choisir le meilleur attribut pour créer un noeud

Le test associé à ce noeud sépare X en deux parties notées X_g et X_d

Construire-arbre(X_g)

Construire-arbre(X_d)

fin si

« vote » des données concernées ou par un tirage au sort pondéré par l'importance relative des classes à cette feuille.

Une interprétation probabiliste

Plaçons-nous au cours de cette construction à un noeud auquel sont attachés n points de l'échantillon d'apprentissage, répartis en C classes ω_j comportant chacune n_j points ($\sum_{j=1,C} n_j = n$).

Considérons un attribut binaire a , dont l'indice n'a pas besoin d'être précisé. Il partage chaque sous-ensemble n_j en deux parties, comportant respectivement l_j et r_j points pour test sur $a = VRAI$ et test sur $a = FAUX$.

Notons :

$$l = \sum_{j=1}^C l_j \quad \text{et} \quad r = \sum_{j=1}^C r_j \quad \text{avec :} \quad r + l = n \quad (11.1)$$

On peut considérer que les n points d'apprentissage sont des tirages aléatoires selon deux distributions discrètes possibles : celle des C valeurs que prend la valeur ω de la classe et celle des deux valeurs de a . On en déduit que :

- l_j/n et r_j/n sont des estimations des probabilités $P(a = VRAI, \omega = \omega_j)$ et $P(a = FAUX, \omega = \omega_j)$.
- l/n et r/n sont des estimations de $P(a = VRAI)$ et de $P(a = FAUX)$.
- n_j/n est une estimation de $P(\omega = \omega_j)$.

Une mesure pour choisir l'attribut

La théorie de l'information nous fournit une mesure naturelle de l'homogénéité entre deux distributions de probabilités à valeurs discrètes : l'*information mutuelle*, ou *entropie croisée* ([Cov91]). En notant ω la première variable et a la seconde, \mathcal{D}_ω et \mathcal{D}_a les ensembles finis des valeurs qu'elles peuvent prendre, l'entropie croisée de ω et de a est donnée par la formule⁷:

$$I(\omega, a) = - \sum_{u,v \in \mathcal{D}_\omega \times \mathcal{D}_a} p(u, v) \log \frac{p(u, v)}{p(u)p(v)} \quad (11.2)$$

7. Dans tout ce chapitre, la base des logarithmes est prise à deux: $\log(a)$ doit se lire comme $\log_2(a)$.

$I(\omega, \mathbf{a})$ présente un minimum à 0 quand, sur tout le domaine $\mathcal{D}_\omega \times \mathcal{D}_{\mathbf{a}}$, on a: $p(u, v) = p(u)p(v)$, c'est-à-dire quand les deux distributions sont indépendantes⁸; elle est en revanche maximale quand les distributions sont complètement corrélées.

La variable aléatoire ω possède une *entropie* $H(\omega)$ qui se définit par:

$$H(\omega) = - \sum_{u \in \mathcal{D}_\omega} p(u) \log(p(u))$$

De même, on peut définir l'entropie de ω *conditionnée par \mathbf{a}* comme:

$$H(\omega | \mathbf{a}) = - \sum_{u, v \in \mathcal{D}_\omega \times \mathcal{D}_{\mathbf{a}}} p(u, v) \log(p(u|v))$$

Un résultat classique de théorie de l'information ([Cov91]) nous affirme alors que:

$$I(\omega, \mathbf{a}) = H(\omega) - H(\omega | \mathbf{a})$$

Dans le cas que nous traitons ici, la variable \mathbf{a} est un attribut binaire, donc $\mathcal{D}_{\mathbf{a}} = \{VRAI, FAUX\}$ et ω représente la distribution des données sur les C classes.

Compte tenu de ce qui a été dit plus haut, les valeurs $H(\omega)$, $H(\omega | \mathbf{a})$ et $I(\omega, \mathbf{a})$ peuvent donc s'estimer par :

$$\begin{aligned} \widehat{I}(\omega, \mathbf{a}) &= - \sum_{j=1}^C \frac{l_j}{n} \log \frac{l_j/n}{(l/n) \times (n_j/n)} + \frac{r_j}{n} \log \frac{r_j/n}{(r/n) \times (n_j/n)} \\ \widehat{I}(\omega) &= - \sum_{j=1}^C \frac{l_j}{n} \log \frac{l_j}{n} \\ \widehat{H}(\omega | \mathbf{a}) &= - \sum_{j=1}^C \frac{l_j}{l} \log \frac{l_j}{l} + \frac{r_j}{r} \log \frac{r_j}{r} \end{aligned}$$

Et on peut vérifier que:

$$\widehat{I}(\omega, \mathbf{a}) = \widehat{I}(\omega) - \widehat{H}(\omega | \mathbf{a})$$

Pour faciliter les calculs, on note:

$$J(\mathbf{a} = VRAI) = \sum_{j=1}^C \frac{l_j}{l} \log \frac{l_j}{l} \quad \text{et} \quad J(\mathbf{a} = FAUX) = \sum_{j=1}^C \frac{r_j}{r} \log \frac{r_j}{r}$$

et donc :

$$\widehat{H}(\omega | \mathbf{a}) = \frac{l}{n} J(\mathbf{a} = VRAI) + \frac{r}{n} J(\mathbf{a} = FAUX) \tag{11.3}$$

Pour construire un noeud dans l'arbre, une idée naturelle et interprétable en terme de théorie de l'information est donc de chercher parmi les d attributs celui qui possède la plus grande corrélation avec la répartition en classes, autrement dit celui qui a la meilleure entropie croisée avec la distribution des points d'apprentissage sur les classes.

8. $0 \log 0$ est pris égal à 0.

Par conséquent, chercher parmi tous les attributs celui qui possède l'information mutuelle la plus grande avec la distribution en classes des n points d'apprentissage revient à trouver celui qui minimise la quantité $\widehat{H}(\omega | \mathbf{a})$, ou, si l'on préfère, à rechercher l'attribut d'indice $i^* \in \{1, d\}$ tel que :

$$i^* = \operatorname{ArgMin}_{i=1,d} \widehat{H}(\omega | \mathbf{a}_i) \quad (11.4)$$

D'autres mesures pour choisir l'attribut

L'entropie croisée n'est pas le seul critère à pouvoir être utilisé : on l'a en effet interprétée comme une mesure de distance entre deux distributions de probabilités. Pourquoi ne pas employer d'autres telles distances, en quittant le strict cadre de la théorie de l'information ? Par exemple, la métrique de Gini ([Gin38]) est très employée en pratique. Son estimation se calcule comme suit, dans les mêmes notations que précédemment :

$$Gini(\omega | \mathbf{a}) = \sum_{i=1}^C \frac{1}{l} (l_j^2 - (ll_j)^2 + \frac{1}{r} (r_j^2 - (rr_j)^2) \quad (11.5)$$

Il existe encore d'autres distances possibles entre distributions de probabilités pouvant servir à construire un arbre de décision : citons en particulier le critère du χ^2 et celui de Lerman ([Ler81]). Le premier s'estime par la formule :

$$\chi^2(c | \mathbf{a}) = \sum_{i=1}^C \left(\frac{l_j - (ln_j/n)}{\sqrt{ln_j/n}} \right)^2 + \left(\frac{r_j - (rn_j/n)}{\sqrt{rn_j/n}} \right)^2 \quad (11.6)$$

et le second par :

$$L(\omega | \mathbf{a}) = \frac{s - esp(s^*)}{\sqrt{var(s^*)}} \quad (11.7)$$

Avec :

$$s = \sum_{j=1}^C \frac{1}{2} (l_j(l_j - 1) + r_j(r_j - 1))$$

$$esp(s^*) = \lambda\mu$$

$$var(s^*) = \lambda\mu + \rho\sigma + \theta\chi - \lambda^2\mu^2$$

avec :

$$\begin{aligned} \lambda &= \frac{l(l-1) + r(r-1)}{\sqrt{2n(n-1)}} \\ \mu &= \frac{\sum_{j=1}^C n_j(n_j - 1)}{\sqrt{2n(n-1)}} \\ \sigma &= \frac{l(l-1)(l-2) + r(r-1)(r-2)}{\sqrt{n(n-1)(n-2)}} \\ \rho &= \frac{\sum_{j=1}^C n_j(n_j - 1)(n_j - 2)}{\sqrt{2n(n-1)(n-2)}} \\ \theta &= \frac{(l(l-1) + r(r-1))^2 - 2(l(l-1)(2l-3) + r(r-1)(2r-3))}{\sqrt{n(n-1)(n-2)(n-3)}} \\ \chi &= \frac{(\sum_{j=1}^C n_j(n_j - 1))^2 - 2\sum_{j=1}^C n_j(n_j - 1)(2n_j - 3)}{\sqrt{n(n-1)(n-2)(n-3)}} \end{aligned}$$

11.1.2.3 Un exemple

Dans l'exemple qui suit, le problème d'apprentissage consiste à trouver une règle de décision binaire à partir de huit exemples sur quatre paramètres binaires. Le problème qui se pose à un enfant qui revient de l'école est le suivant : *peut-il aller jouer chez son voisin ou pas?* L'expérience, qu'il a acquise par punition récompense sur les huit jours d'école précédents, est résumée dans le tableau n des huit exemples d'apprentissage suivants :

	mes Devoirs sont-ils Finis ?	Maman est-elle de Bonne Humeur ?	Est-ce qu'il Fait Beau ?	Mon Goûter est-il Pris ?	DÉCISION
1	VRAI	FAUX	VRAI	FAUX	OUI
2	FAUX	VRAI	FAUX	VRAI	OUI
3	VRAI	VRAI	VRAI	FAUX	OUI
4	VRAI	FAUX	VRAI	VRAI	OUI
5	FAUX	VRAI	VRAI	VRAI	NON
6	FAUX	VRAI	FAUX	FAUX	NON
7	VRAI	FAUX	FAUX	VRAI	NON
8	VRAI	VRAI	FAUX	FAUX	NON

Pour construire la racine de l'arbre de décision, il faut d'abord trouver l'attribut dont la distribution possède l'entropie mutuelle la plus faible avec celle de la décision.

Notons, pour simplifier, $H(\omega|DF)$ pour $H(\omega|Mes\ Devoirs\ sont-ils\ Finis?)$ et de la même manière: $H(\omega|DF)$, $H(\omega|BH)$, $H(\omega|FB)$, $H(\omega|GP)$.

On a :

$$H(\omega|DF) = \frac{5}{8}J(DF = VRAI) + \frac{3}{8}J(DF = FAUX)$$

avec

$$J(DF = VRAI) = -\frac{3}{5}\log\left(\frac{3}{5}\right) - \frac{2}{5}\log\left(\frac{2}{5}\right)$$

et

$$J(DF = FAUX) = -\frac{1}{3}\log\left(\frac{1}{3}\right) - \frac{2}{3}\log\left(\frac{2}{3}\right)$$

Soit :

$$H(\omega|DF) \approx 0.93$$

On trouve par un calcul analogue :

$$H(\omega|FB) \approx 0.80$$

$$H(\omega|BH) \approx 0.93$$

$$H(\omega|GP) = 1.$$

On choisit donc pour racine de l'arbre le test *Est-ce qu'il Fait Beau?* qui minimise l'entropie croisée avec la distribution en classes.

Sous la branche gauche portant la valeur *VRAI* se trouve le tableau suivant des exemples corrects pour ce test :

	mes Devoirs sont-ils Finis ?	Maman est-elle de Bonne Humeur ?	Mon Goûter est-il Pris ?	DECISION
1	VRAI	FAUX	FAUX	OUI
3	VRAI	VRAI	FAUX	OUI
4	VRAI	FAUX	VRAI	OUI
5	FAUX	VRAI	VRAI	NON

Sous la branche Droite, portant la valeur *FAUX* se trouve le tableau suivant:

	mes Devoirs sont-ils Finis ?	Maman est-elle de Bonne Humeur ?	Mon Goûter est-il Pris ?	DECISION
2	FAUX	VRAI	VRAI	OUI
6	FAUX	VRAI	FAUX	NON
7	VRAI	FAUX	VRAI	NON
8	VRAI	FAUX	FAUX	NON

La poursuite du procédé conduit finalement à l'arbre de décision de la figure 11.2.

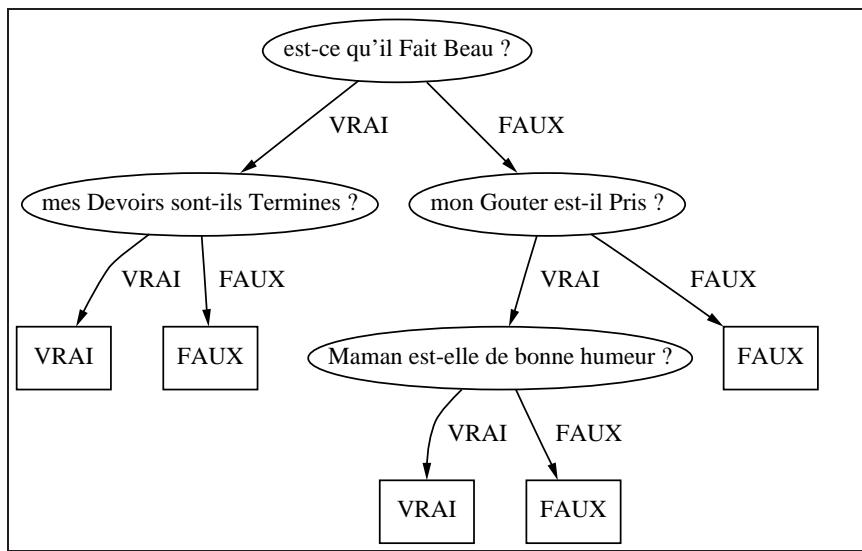


FIG. 11.2 – L’arbre de décision construit sur les huit exemples précédents.

11.1.2.4 Le cas des attributs non binaires

On a traité ci-dessus le cas des attributs binaires. Mais il est possible de calculer de la même manière une entropie croisée avec la distribution en classes pour les autres types d’attributs.

Le cas binaire

Pour mémoire, le test consiste ici à descendre dans un sous arbre si le test sur l’attribut choisi vaut *VRAI*, dans l’autre s’il vaut *FAUX*.

Le cas nominal

Le cas où les attributs sont à valeurs discrètes se généralise facilement quand le test que l’on construit se réduit à opposer une valeur à toutes les autres : on est alors ramené au

cas binaire. Par exemple, s'il existe un attribut **couleur** prenant ses valeurs dans l'ensemble $\{bleu, rouge, vert, jaune\}$, il est simple de l'éclater en quatre attributs binaires, du type **couleur - rouge**, qui est *VRAI* ou *FAUX* sur chaque donnée d'apprentissage. On se replace alors dans le cas exposé ci-dessus, avec la transformation d'un attribut nominal à k valeurs possibles en k attributs binaires que l'on traite indépendamment.

Cette technique a l'inconvénient d'oublier la signification globale de l'attribut; en effet, si **couleur-rouge** est *VRAI* pour un attribut, **couleur-bleu** est automatiquement *FAUX*; mais cette propriété n'apparaît plus explicitement dans les données. Une autre solution est alors de calculer directement l'information mutuelle entre les deux variables à valeurs discrètes que sont d'une part cet attribut et d'autre part l'ensemble des classes. Si celle-ci se révèle la meilleure pour tous les attributs, on crée alors un nœud non binaire dans l'arbre de décision (dans l'exemple précédent, le test de l'attribut « couleur » donne quatre réponses possibles). Le seul inconvénient est qu'il faut gérer une structure de données plus complexe.

Le cas continu

Traiter un attribut continu peut paraître plus difficile, mais en pratique ce n'est pas fondamentalement différent : puisque le nombre de données d'apprentissage est fini, le nombre des valeurs que prend cet attribut sur les exemples est aussi fini. Mieux, ses valeurs sont ordonnées, contrairement au cas nominal. Le sélecteur consistera donc à comparer les valeurs à un seuil pour construire un noeud binaire.

Pour un attribut a continu, on procède alors ainsi: on trie les points d'apprentissage selon la valeur de cet attribut, puis on cherche le seuil $s(a)$ qui minimise l'un des critères précédents⁹.

Il est à noter que l'arbre de décision est le seul modèle permettant de gérer de manière homogène les attributs de nature variés, en particulier les mélanges continus et binaires.

L'utilisation simultanée de plusieurs attributs continus

Dans le cas où un attribut est continu, chaque test sur cet attribut n'est autre que la comparaison à un seuil; si tous les attributs sont continus, on obtient donc finalement dans \mathbb{R}^d des surfaces de séparation entre les régions attribuées aux classes qui sont composées d'hyperplans *orthogonaux aux axes*. Il est tentant de relâcher cette contrainte, pour éviter de construire des arbres complexes sur des situations simples, comme sur la figure 11.10. On peut alors chercher des séparatrices linéaires non parallèles aux axes en utilisant des tests sur des combinaisons linéaires d'attributs à chaque nœud, et non sur un seul attribut. On réalise alors un arbre de décision *oblique*.

Cependant, la méthode proposée pour un attribut continu ne peut pas se généraliser pour la combinaison d'attributs continus : l'espace de recherche est cette fois infini, puisque l'on cherche à chaque nœud des valeurs (en nombre $d+1$) non contraintes comme précédemment par une relation d'ordre. On emploie alors des techniques d'optimisation, comparables à celles du chapitre 9. Les méthodes présentées dans [MKS94] et [BU92] sont des exemples efficaces ; on y reviendra sur un exemple au paragraphe 11.1.4.

11.1.3 Comment élaguer un arbre trop précis

11.1.3.1 Pourquoi élaguer ?

On a vu que la poursuite de l'algorithme de construction jusqu'à son terme naturel fabrique un arbre T_{max} dont les feuilles sont pures, c'est-à-dire correspondent à des exemples de la même classe; il y a là clairement un risque de mésestimation de la probabilité d'erreur par le taux

9. Ceci nécessite, pour les n données, l'examen de $n - 1$ seuils : par exemple les valeurs médianes entre deux points d'apprentissage dans leur liste triée.

d'erreur apparent, qui vaut ici exactement 0. On se trouve donc dans le cas exposé au chapitre 2 : le nombre de nœuds de l'arbre de décision est un critère de complexité simple et efficace pour lequel les courbes présentées à la figure 3.13 dans le chapitre 3 sont caractéristiques. Chercher la valeur « optimale » k_0 du nombre de nœuds revient donc à trouver une technique pour contrôler la taille de l'arbre. Il s'agit donc d'une méthode de régularisation ou de sélection de modèle (voir chapitre 2).

11.1.3.2 Une première solution : le préélagage

Une solution simple consiste à cesser de diviser un nœud quand la pureté des points qu'il domine est non pas parfaite, mais suffisante. Une fois sélectionné le meilleur attribut, on regarde si la valeur du critère de la division est inférieure à un certain seuil ; en pratique, ceci revient à admettre que, s'il existe une classe suffisamment majoritaire sous un nœud, on peut considérer ce dernier comme une feuille et lui attribuer la classe en question. Selon le critère de division utilisé, diverses heuristiques ont été proposées pour régler le seuil précédent. Sa valeur peut d'ailleurs être variable selon le nœud où l'on se trouve, dépendant de l'estimation de la probabilité *a priori* des classes, de l'estimation empirique de la difficulté à les séparer, etc.

Ces méthodes présentent certains inconvénients, dont le principal est qu'elles sont myopes (puisque ne prennent en compte qu'un critère local à la feuille examinée), et peuvent de ce fait manquer un développement de l'arbre qui serait excellent. C'est pourquoi on leur préfère souvent des méthodes d'élagage *a posteriori*, une fois que l'arbre a été entièrement développé.

11.1.3.3 Le post-élagage par un ensemble indépendant de validation

Une autre technique, plus valide théoriquement et plus efficace en pratique, consiste à d'abord construire l'arbre de décision complètement, puis seulement après à chercher à le simplifier en l'élaguant progressivement en remontant des feuilles vers la racine. Pour juger quand il est bon d'arrêter d'élaguer l'arbre, on utilise un critère de qualité qui exprime souvent un compromis entre l'erreur commise par l'arbre et une mesure de sa complexité.

L'erreur commise est mesurée grâce à un ensemble de validation (voir chapitre 3). On supposera donc dans ce paragraphe que l'ensemble d'apprentissage est assez important pour être coupé en deux parties : l'une (ensemble d'apprentissage proprement dit) pour construire l'arbre de décision T_{max} , l'autre (ensemble de validation) pour choisir le meilleur parmi les élagages proposés.

L'algorithme optimal consisterait à calculer le taux d'erreur de l'ensemble de validation sur tous les arbres qu'il est possible d'obtenir par élagage de T_{max} . Mais leur nombre croît très rapidement avec la taille de T_{max} , mesurée en nombre de nœuds¹⁰. On utilise donc des solutions sous-optimales, dont la plus classique (un algorithme *glouton*) consiste à construire sans retour en arrière une séquence d'arbres par élagages successifs, en remontant des feuilles vers la racine.

Cette séquence se note $S = (T_{max}, T_1, \dots, T_k, \dots, T_n)$. T_n est l'arbre constitué d'une seule feuille comprenant les m points d'apprentissage. C'est donc l'arbre élagué au maximum. Pour passer de T_k à T_{k+1} , il faut transformer un nœud dans T_k en feuille. Pour savoir si cet élagage serait bénéfique, l'idée générale est de comparer le « coût » de l'arbre élagué et celui de l'arbre non élagué, et d'arrêter l'élagage quand le coût du premier dépasse le coût du second. Pour évaluer ce coût, plusieurs critères ont été proposés qui prennent tous en compte à la fois l'erreur commise par l'arbre et une mesure de sa complexité (voir en particulier les articles de synthèse [BA97, EMS97, Min89]).

10. Ou en nombre de feuilles, car un arbre binaire T ayant $|T|$ nœuds (feuilles comprises) possède exactement $(|T|-1)/2$ feuilles.

Nous examinons ici le critère consistant à choisir le nœud ν qui minimise sur l'ensemble des nœuds de T_k la valeur suivante :

$$\varpi(T_k, \nu) = \frac{MC_{éla}(\nu, k) - MC(\nu, k)}{n(k).(nt(\nu, k) - 1)} \quad (11.8)$$

où :

- $MC_{éla}(\nu, k)$ est le nombre d'exemples de l'ensemble d'apprentissage mal classés par le nœud ν de T_k dans l'arbre élagué à ν .
- $MC(\nu, k)$ est le nombre d'exemples de l'ensemble d'apprentissage mal classés sous le nœud ν dans l'arbre non élagué
- $n(k)$ est le nombre de feuilles de T_k
- $nt(\nu, k)$ est le nombre de feuilles du sous-arbre de T_k situé sous le nœud ν .

Ce critère permet donc d'élaguer un nœud de T_k de façon à ce que T_{k+1} , l'arbre obtenu, possède le meilleur compromis entre taille et taux d'erreur apparent.

Finalement, la suite $S = (T_{max}, T_1, \dots, T_k, \dots, T_n)$ possède un élément T_{k_0} pour lequel le nombre d'erreurs commises est minimal sur l'ensemble de validation : c'est cet arbre-là qui sera finalement retenu par la procédure d'élagage.

Algorithme 11.2 Elagage d'un arbre de décision

```

Procédure : élaguer( $T_{max}$ )
   $k \leftarrow 0$ 
   $T_k \leftarrow T_{max}$ 
  tant que  $T_k$  a plus d'un nœud faire
    pour chaque nœud  $\nu$  de  $T_k$  faire
      calculer le critère  $\varpi(T_k, \nu)$  sur l'ensemble d'apprentissage
    fin pour
    choisir le nœud  $\nu_m$  pour lequel le critère est maximum
     $T_{k+1}$  se déduit de  $T_k$  en y remplaçant  $\nu_m$  par une feuille
     $k \leftarrow k + 1$ 
  fin tant que

```

Dans l'ensemble des arbres $\{T_{max}, T_1, \dots, T_k, \dots, T_n\}$, choisir celui qui a la plus petite erreur de classification sur l'ensemble de validation.

11.1.3.4 Un exemple d'élagage

Les figures 11.3 et 11.4 représentent un petit ensemble d'exemples à deux classes et deux attributs numériques, ainsi que l'arbre de décision T_{max} appris par l'algorithme donné ci-dessus.

En appellant ν_1 le nœud racine de T_{max} , ν_2 et ν_3 ses fils gauche et droit et ν_4 son dernier nœud intérieur (le fils gauche de ν_2), on peut calculer les valeurs :

$$\begin{aligned} \varpi(T_{max}, \nu_1) &= \frac{MC_{éla}(\nu_1, k) - MC(\nu_1, k)}{n(k).(nt(\nu_1, k) - 1)} = \frac{9 - 0}{5.(5 - 1)} = 9/20 \\ \varpi(T_{max}, \nu_2) &= \frac{1 - 0}{5.(3 - 1)} = 1/10 \end{aligned}$$

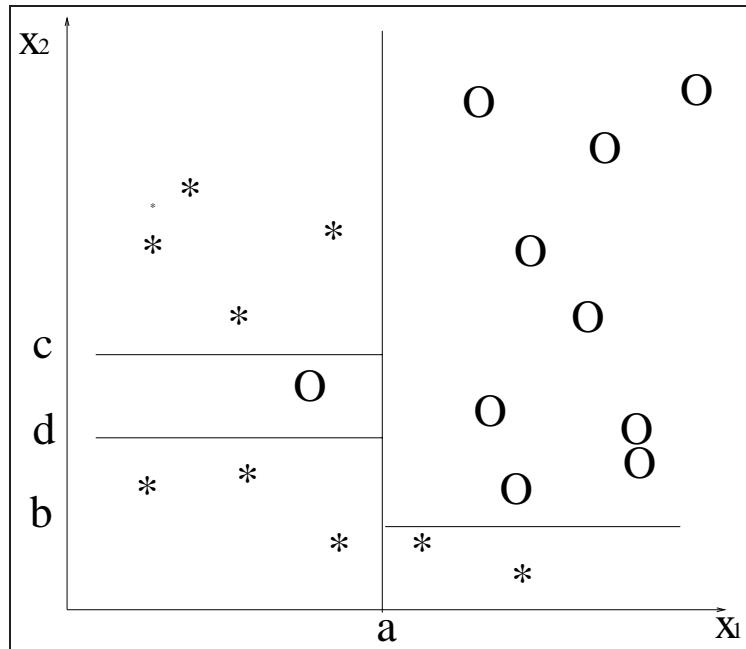
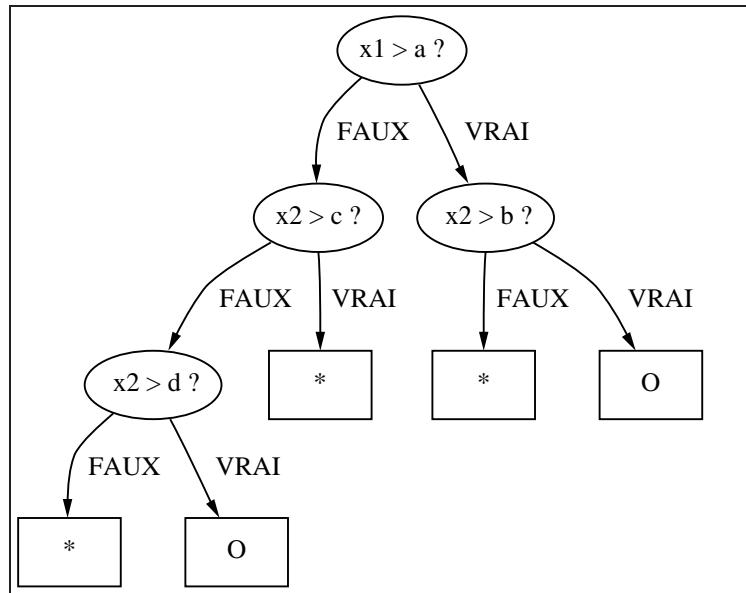


FIG. 11.3 – L’arbre de décision géométrique.

FIG. 11.4 – L’arbre de décision logique T_{max} .

$$\varpi(T_{max}, \nu_3) = \frac{1 - 0}{5 \cdot (2 - 1)} = 1/5$$

$$\varpi(T_{max}, \nu_4) = \frac{1 - 0}{5 \cdot (2 - 1)} = 1/5$$

Par conséquent, l’arbre T_1 sera le résultat de l’élagage de T_{max} au nœud ν_2 , soit celui de la figure 11.5.

En travaillant désormais sur T_1 , on trouve les valeurs :

$$\varpi(T_1, \nu_1) = \frac{9 - 1}{3 \cdot (3 - 1)} = 4/3$$

$$\varpi(T_1, \nu_3) = \frac{2 - 0}{3 \cdot (3 - 1)} = 2/3$$

L'arbre T_2 choisi résultera de l'élagage de ν_3 dans T_1 ; il aura donc pour seul nœud la racine de T_{max} , avec une feuille pour chaque classe.

Puisque l'on suppose disposer d'un ensemble de validation, c'est en testant ce dernier sur les arbres T_{max} , T_1 et T_2 que l'on vient de calculer¹¹ que l'on choisira celui qui possède la meilleure estimation de taux d'erreur de classification. La procédure d'élagage est alors terminée.

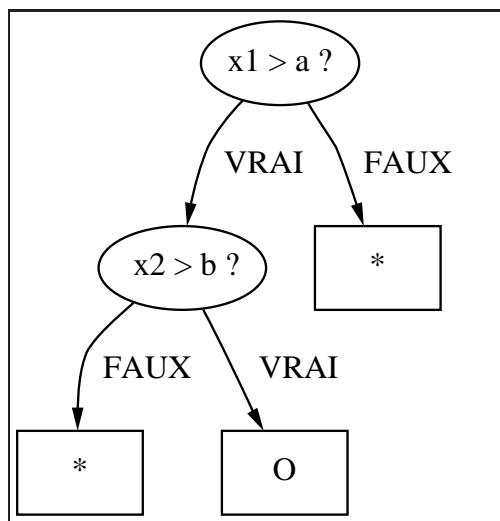


FIG. 11.5 – Un arbre de décision logique après un premier élagage : T_1 .

11.1.4 Un exemple : les iris de Fisher

Les figures 11.6, 11.7, 11.8 et 11.9 illustrent les différents points présentés. Elles ont été créées à partir du logiciel OC1 ([MKS94]). Les données présentées sont celles des iris de Fisher¹². Le problème est de classer les iris en trois classes connaissant un certain nombre de leurs caractéristiques.

Les attributs présentés ici sont la longueur et la largeur des sépales de trois classes différentes d'Iris, sur cent cinquante exemples. Les trois classes sont référencées dans l'ensemble $\{1, 2, 3\}$.

Les données ont été partagées aléatoirement en deux parties, l'une d'apprentissage pour la construction et l'élagage de T_{max} , l'autre de test pour l'estimation de taux d'erreur de la classification ainsi obtenue. Il y a cent exemples d'apprentissage et cinquante de test.

La convention graphique de représentation d'un arbre de classification dans un espace à deux dimensions est la suivante: la droite notée **Root** est le premier test ; par exemple, dans

11. Plus l'arbre noté plus haut T_N , composé d'une seule feuille et qui ne représente que la probabilité *a priori* des classes dans l'ensemble d'apprentissage

12. Disponibles sur le site de l'université de Californie à Irvine (UCI) :
<http://www.ics.uci.edu/>

la figure 11.6, elle représente un test sur la première coordonnée et est donc parallèle à l'axe vertical. Les droites notées r et l représentent les tests faits juste ensuite; les suivants sont notés rl et rr , lr et ll , et ainsi de suite. La notation r signifie donc « fils droit », l « fils gauche ».

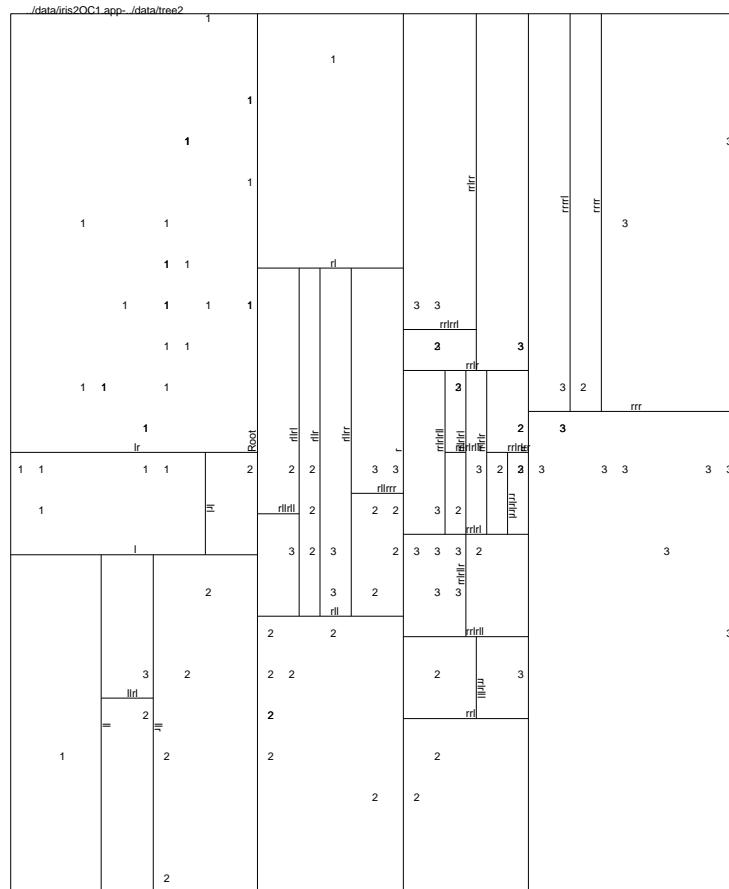


FIG. 11.6 – Les cent données d'apprentissage sur l'arbre non élagué.

On voit dans la figure 11.6 un arbre T_{max} représenté avec les cent données d'apprentissage qui ont servi à le construire. On peut s'attendre à ce qu'il sépare parfaitement les trois classes; en réalité, à cause des points de classes différentes et de mêmes coordonnées, quelques choix arbitraires sont faits. Le taux d'erreur apparent n'est donc pas nul, mais vaut 2 %. Le nombre de tests maximal est de dix (il correspond au segment de droite noté $rrrlrlrlrlr$).

On a représenté ensuite les données de test sur T_{max} : elles produisent une estimation de l'erreur de classification valant 32 %.

La version standard de OC1 avec élagage met de côté un dixième des données d'apprentissage comme ensemble de validation. La figure 11.8 représente les quatre-vingt-dix données d'apprentissage restantes séparées par l'arbre élagué grâce à ces dix données (à partir de l'arbre T_{max} construit sur ces mêmes quatre-vingt-dix données)¹³. Le taux d'erreur apparent est monté à 25 %. L'arbre élagué est de profondeur 2. Il est constitué de deux sélecteurs sur la même coordonnée.

13. Cet arbre T_{max} est différent du précédent, puisqu'il est construit sur quatre-vingt-dix données tirées aléatoirement et non pas cent.

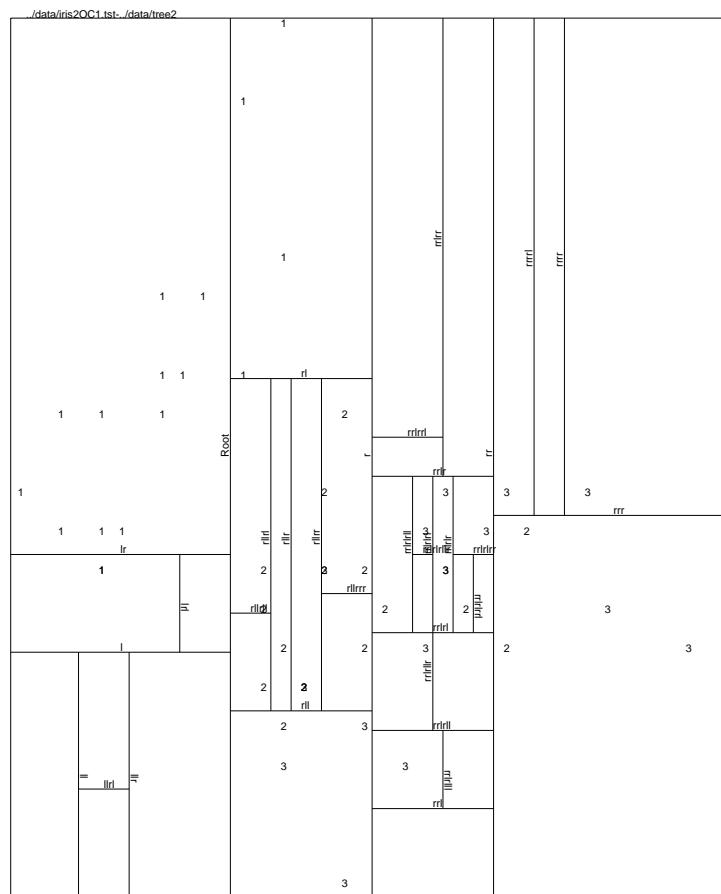


FIG. 11.7 – Les cinquante données de test sur l’arbre non élagué.

Pour finir, les cinquante données de test sont confrontées à l’arbre élagué (Figure 11.9) ce qui donne une estimation de l’erreur de classification de 26 %.

Bien que les deux dernières estimations présentent un intervalle de confiance large, elle correspondent cependant à l’attente : le surapprentissage est bel et bien corrigé par l’élagage.

Le logiciel OC1 permet également de construire des arbres de décision sur des données numériques par combinaison linéaire des attributs ; autrement dit, on obtient dans ce cas des droites séparatrices non parallèles aux axes.

La figure 11.10 montre comment un « arbre oblique » de profondeur maximale 2 permet de séparer les cinquante données de test. Il est construit sur quatre-vingt-dix données d’apprentissage et élagué sur dix données de validation. L’estimation de l’erreur de classification est de 24 %.

11.1.5 Traduction des arbres de décision en logique des propositions

Dans l’optique du chapitre 3, on peut voir les arbres de décision comme la construction imbriquée de sélecteurs et de règles de généralisation en logique des propositions. Chaque branche de l’arbre correspond à une conjonction de tests associés à une classe. L’ensemble des branches peut donc être considéré comme une disjonction exhaustive et exclusive de conjonctions (tous les exemples possibles sont couverts chacun par une règle et une seule). Nous l’illustrons sur un exemple.

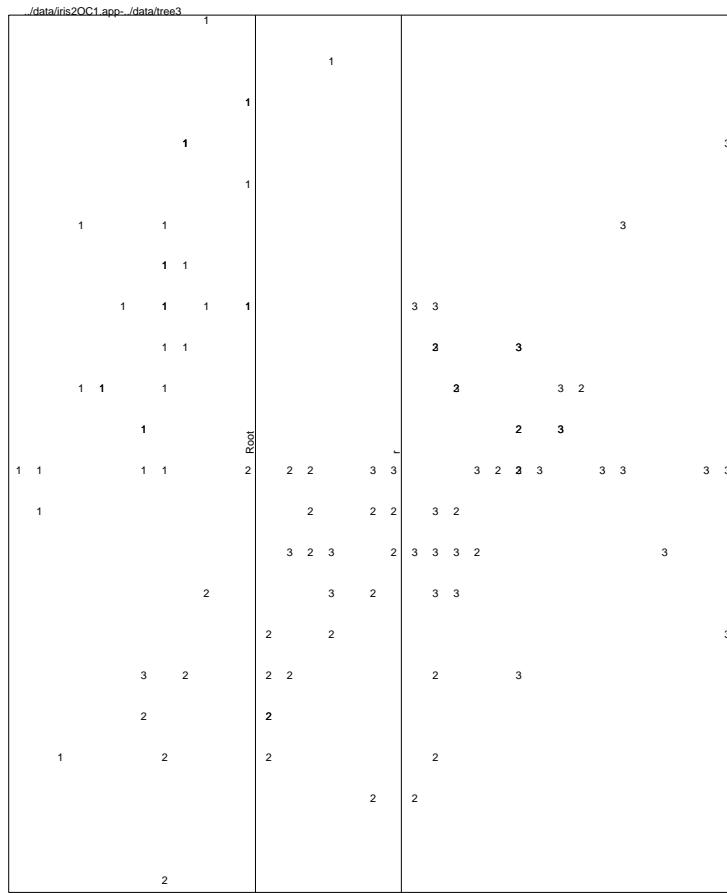


FIG. 11.8 – Les quatre-vingt-dix données d'apprentissage restantes sur l'arbre élagué par dix données.

Supposons les données d'apprentissage décrites par trois variables: la première à valeur continue: **fièvre**, la seconde nominale: **qualité**, pouvant prendre une des trois valeurs **homme**, **femme**, **enfant** et la troisième binaire: **réaction positive ou négative au test T**. La classification à effectuer porte sur le diagnostic d'un certain syndrome **S**. Supposons que l'algorithme de construction ait calculé l'arbre suivant à partir des données d'apprentissage:

```

Si qualité = enfant
Alors      S = FAUX
Sinon:    Si réaction négative à T
          Alors      S = FAUX
          Sinon      Si qualité = femme
                      Alors      S = VRAI
                      Sinon      Si fièvre ≥ 39°
                                  Alors S = VRAI
                                  Sinon S = FAUX

```

Le concept appris, représenté maintenant en logique des propositions, peut se décrire ainsi: l'algorithme d'apprentissage a créé trois sélecteurs pour définir sa nouvelle représentation des

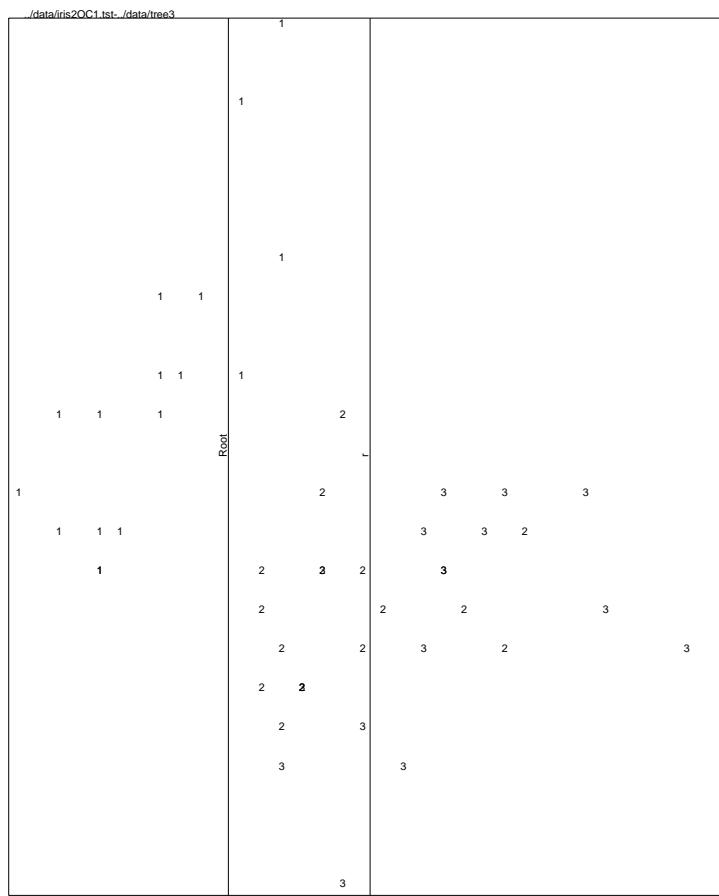


FIG. 11.9 – Les cinquante données de test restantes sur l’arbre élagué.

connaissances :

- Un seuil à 39° , qui permet de binariser l’utilisation de la variable continue **fièvre** (remarquons qu’en général, plusieurs seuils peuvent apparaître dans les variables continues). Notons a_1 le fait qu’un patient ait une fièvre supérieure à 39° , \bar{a}_1 le contraire.
- La transformation de la variable nominale **qualité** en trois variables binaires **enfant**, **homme**, **femme**, que nous notons a_2 , a_3 et a_4 . Il est à remarquer que désormais, le fait que deux d’entre elles ne puissent pas être vraies à la fois apparaîtra implicitement.
- La variable binaire **réaction au test T**, qui reste binaire: notons-la a_5 .

Finalement, l’apprentissage réalisé par la construction de l’arbre de décision ci-dessus peut se traduire dans la nouvelle représentation des connaissances par la découverte du concept :

$$(\bar{a}_2 \wedge a_5 \wedge a_3) \vee (\bar{a}_2 \wedge a_5 \wedge \bar{a}_3 \wedge a_1)$$

D’une manière générale, un arbre de décision est une représentation de l’apprentissage d’un concept sous la forme d’une *disjonction de conjonctions*.

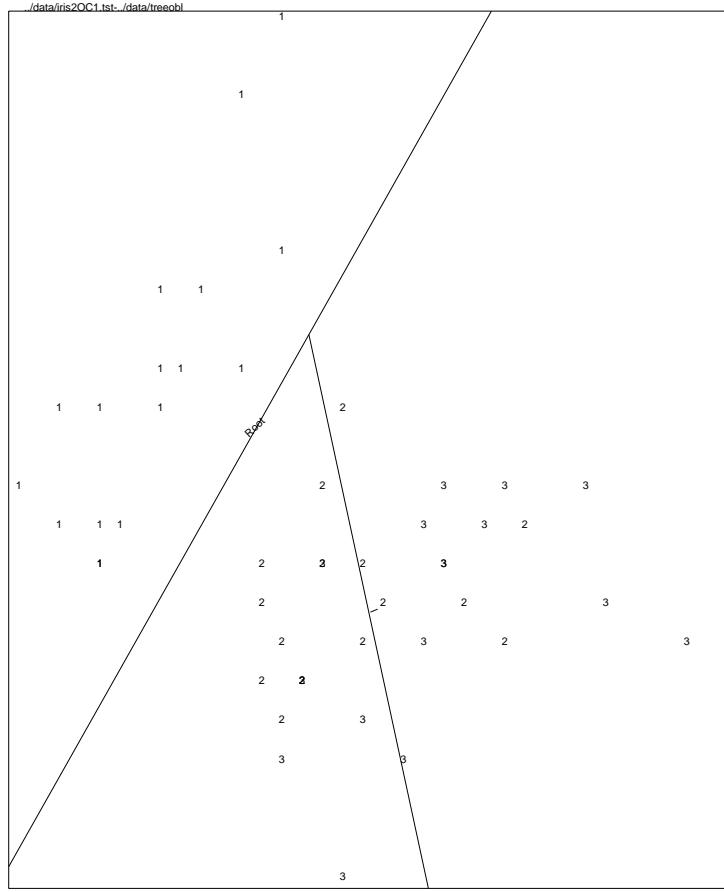


FIG. 11.10 – Les 50 données de test restantes sur l’arbre oblique élagué.

11.2 Les arbres de régression

11.2.1 Le principe

Nous avons vu, par exemple sur la figure 11.3, que les arbres de classification découpent l'espace des entrées $\mathbb{R}^d = \{x_1, \dots, x_i, \dots, x_d\}$ en régions dont les côtés sont des hyperplans perpendiculaires aux axes. À l'intérieur de chacune de ces régions, la valeur prédictée est constante : c'est une des classes ω_j , pour $i = 1, C$. Les arbres de régression utilisent le même mécanisme, à ceci près que les valeurs à prédire sont continues : après apprentissage sur un ensemble $\mathcal{S} = \{\mathbf{x}_1, \dots, \mathbf{x}_m\}$, on saura associer à chaque objet $x \in \mathbb{R}^d$ une valeur réelle de sortie y . Les arbres de régression prévoient une valeur numérique c_k constante pour chaque région construite après l'apprentissage et l'élagage. En supposant que le modèle induit contienne M régions R_1, \dots, R_M et en notant $\mathcal{I}(R_k)$ la fonction caractéristique de la région R_k , qui vaut 1 pour les points appartenant à R_k et 0 ailleurs, la fonction de sortie y associée à ce modèle est donc :

$$y = \sum_{k=1}^M c_k \mathcal{I}(R_k)$$

Nous considérons seulement le mécanisme le plus utilisé, celui de la régression quadratique. Il construit l'arbre en tentant de minimiser le carré de la différence entre les valeurs observées

et valeurs prévues. Pour les m exemples d'apprentissage, il faut donc idéalement minimiser, en notant Δ la distance euclidienne:

$$\sum_{i=1}^m \Delta^2(\mathbf{u}_i - \mathbf{y}_i)$$

En considérant maintenant une optimisation région par région, cette expression est minimale pour la région R_n contenant n entrées lorsqu'elle est égale à la moyenne g des valeurs des points d'apprentissage qu'elle contient (voir une démonstration par exemple au chapitre 14).

11.2.2 La construction

L'algorithme utilisé pour construire un arbre de régression est glouton, comme celui de la construction d'un arbre de décision. L'attribut choisi à l'étape courante est celui qui peut se discréteriser par un découpage en deux minimisant le critère de la somme des écarts à la moyenne dans chaque partie du découpage. Ainsi, pour chaque attribut x_k et pour chaque valeur de découpage d_{jk} de cet attribut, on définit deux régions séparées par cette valeur et on obtient deux valeurs du critère que nous notons C_{jk}^G et C_{jk}^D .

On choisit au total d'utiliser la variable x_k et le découpage d_{jk} qui minimisent la somme des deux valeurs ci-dessus.

11.2.3 Un exemple

Voyons une étape de la construction d'un arbre de régression à deux dimensions. Les données d'apprentissage sont au nombre de seize et le découpage courant est celui donné à la figure 11.11.

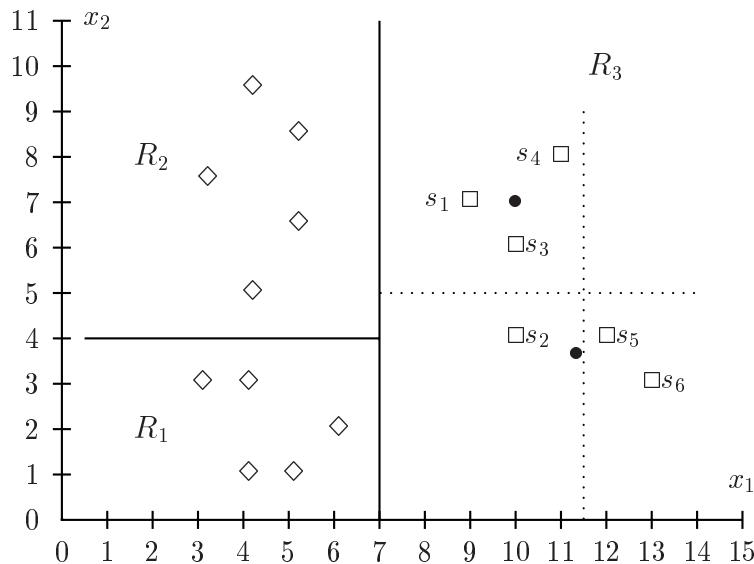


FIG. 11.11 – Une étape de la construction d'un arbre de régression.

Continuons la construction par la séparation en deux de la région R_3 . Les valeurs de seuil possibles pour x_1 sont celles qui passent au milieu de deux points de R_3 , c'est-à-dire les quatre valeurs 9.5, 10.5, 11.5 et 12.5. La troisième correspond à la droite verticale qui a été tracée en pointillés.

Quelle est la valeur du critère pour cette hypothèse? Elles sépare les points de \mathcal{R}_3 en deux groupes: $\{s_1, s_2, s_3, s_4\}$ et $\{s_5, s_6\}$. Le centre de gravité (la moyenne) g_1 du premier groupe est aux coordonnées (10, 5.75) et son homologue g_2 pour le second groupe est aux coordonnées (12.5, 3.5). Le critère vaut donc:

$$\mathcal{C}_{31}^G + \mathcal{C}_{31}^D = \sum_{i \in \{1, 2, 3, 4\}} \Delta^2(s_i - g_1) + \sum_{i \in \{5, 6\}} \Delta^2(s_i - g_2) \simeq 11.85$$

Il faut aussi s'intéresser à x_2 . Quatre valeurs de seuil sont possibles pour cet attribut: 3.5, 5, 6.5, 7.5. La droite correspondant à la seconde valeur 5 a été tracée en pointillés. Elle sépare les points de \mathcal{R}_3 en deux groupes: $\{s_4, s_1, s_3\}$ et $\{s_2, s_5, s_6\}$. Le centre de gravité h_1 du premier est aux coordonnées (10, 7) et h_2 est aux coordonnées (11.3, 3.7) (ces deux points sont indiqués sur la figure par le symbole \bullet). Le critère vaut ici:

$$\mathcal{C}_{22}^G + \mathcal{C}_{22}^D = \sum_{i \in \{4, 1, 3\}} \Delta^2(s_i - h_1) + \sum_{i \in \{2, 5, 6\}} \Delta^2(s_i - h_2) \simeq 9$$

Le calcul complet montrerait que parmi toutes les séparations possibles sur x_1 et x_2 , c'est cette dernière qui est la meilleure du point de vue du critère quadratique employé. La région R_3 sera donc divisée en deux par la droite d'équation $x_2 = 5$. Il est intéressant de noter que ce n'est pas la médiatrice entre h_1 et h_2 .

11.2.4 La fin de la construction et l'élagage

Cette construction se poursuit jusqu'à ce que chaque point soit sur une feuille, ou lorsque les moyennes des deux régions les meilleures à séparer sont trop proches. Cette dernière façon de faire est cependant dangereuse dans la mesure où un attribut très séparateur peut succéder à un qui ne l'est pas.

C'est pourquoi on a développé pour les arbres de régression des méthodes d'élagage puissantes. Après avoir laissé croître l'arbre jusqu'à ce que chaque feuille ne contienne qu'un petit nombre de points, voire un seul, on élague en réunissant les feuilles selon un critère de complexité dont Breiman [BFOS84] a montré qu'il est optimal pour un arbre donné. En d'autres termes, la procédure d'élagage ne transforme pas un arbre sous-optimal en un arbre optimal, bien entendu, elle se contente d'être capable, pour un arbre donné, de trouver l'élagage optimal pour cet arbre.

11.3 Le *boosting* d'un algorithme d'apprentissage

11.3.1 Plusieurs experts valent mieux qu'un

Il est rare qu'un décideur ait sous la main un expert omniscient et incontesté lui permettant de faire le meilleur choix. Il n'a souvent d'autres ressources que de consulter un comité d'experts plus ou moins compétents puis de combiner leurs avis pour prendre sa décision. Mais cette décision est-elle forcément la bonne? Est-elle meilleure que la décision qu'aurait pris le meilleur expert du comité? Peut-on s'arranger pour rendre ce comité d'experts de plus en plus performant?

Prenons un exemple concret¹⁴. Soit un joueur de tiercé cherchant à maximiser ses gains. Il connaît un certain nombre d'« experts » des courses de chevaux. Aucun d'eux n'est capable d'expliquer complètement son expertise, mais, interrogé à propos d'un ensemble de courses,

14. Repris de Freund et Schapire, les concepteurs du boosting, dans [FS99].

chacun d'eux peut fournir des règles grossières (par exemple: « il faut parier sur le cheval ayant gagné le plus grand nombre de courses », ou: « il faut parier sur le cheval ayant la plus grande cote »). Prise isolément, chacune de ces règles est peu performante. On peut cependant raisonnablement penser qu'elles sont un peu meilleures que le hasard. De plus, si on interroge chaque expert sur des ensembles de courses différents, on peut obtenir plusieurs règles de ce type. Le joueur a maintenant deux questions à résoudre. D'abord, quels ensembles de courses devrait-il présenter à chaque expert en vue d'extraire les règles les plus intéressantes ? Ensuite, comment doit-il combiner les avis des experts pour atteindre la meilleure décision ? La première question concerne le choix des exemples d'apprentissage soumis à l'apprenant. La deuxième concerne la manière de combiner l'avis d'apprenants (potentiellement différents) entraînés sur des échantillons différents.

De manière étonnante, des recherches en apprentissage artificiel datant du début des années 1990 montrent qu'il est possible d'atteindre une décision aussi précise que souhaitée par une combinaison judicieuse d'experts imparfaits mais correctement entraînés. Plusieurs algorithmes d'apprentissage ont été développés à la suite de ces travaux.

Le mot *boosting*¹⁵ s'applique à des méthodes générales capables de produire des décisions très précises (au sens d'une fonction de perte) à partir d'un ensemble de règles de décision « faibles », c'est-à-dire dont la seule garantie est qu'elles soient un peu meilleures que le hasard. Ces méthodes s'appliquent aussi bien à l'estimation de densité qu'à la régression ou à la classification. Pour simplifier, nous nous concentrerons ici sur la tâche de classification binaire.

Dans sa version « par sous-ensembles », cette technique fait produire à l'algorithme trois résultats selon la partie de l'ensemble d'apprentissage sur laquelle il apprend, puis combine les trois apprentissages réalisés pour fournir une règle de classification plus efficace. Examinons d'abord cette technique avant de voir comment la généraliser à l'aide de distributions de probabilité sur les exemples.

11.3.2 Le premier algorithme de boosting

Schapire ([Sch90]) développa le premier algorithme de boosting pour répondre à une question de Kearns : est-il possible de rendre aussi bon que l'on veut un algorithme d'apprentissage « faible », c'est-à-dire un peu meilleur que le hasard ? Shapire montra qu'un algorithme faible peut toujours améliorer sa performance en étant entraîné sur trois échantillons d'apprentissage bien choisis. Nous ne nous intéressons ici qu'à des problèmes de classification binaire.

L'idée est d'utiliser un algorithme d'apprentissage qui peut être de natures très diverses (un arbre de décision, une règle bayésienne de classification, une décision dépendant d'un hyperplan, etc.) sur trois sous-ensembles d'apprentissage.

1. On obtient d'abord une première hypothèse h_1 sur un sous-échantillon \mathcal{S}_1 d'apprentissage de taille $m_1 < m$ (m étant la taille de \mathcal{S} l'échantillon d'apprentissage disponible).
2. On apprend alors une deuxième hypothèse h_2 sur un échantillon \mathcal{S}_2 de taille m_2 choisi dans $\mathcal{S} - \mathcal{S}_1$ dont la moitié des exemples sont mal classés par h_1 .
3. On apprend finalement une troisième hypothèse h_3 sur m_3 exemples tirés dans $\mathcal{S} - \mathcal{S}_1 - \mathcal{S}_2$ pour lesquels h_1 et h_2 sont en désaccord.
4. L'hypothèse finale est obtenue par un vote majoritaire des trois hypothèses apprises :

$$H = \text{vote majoritaire}(h_1, h_2, h_3)$$

15. La traduction littérale de ce mot est « stimulation » ou « amplification » (pourquoi pas « dopage »?) ; le terme anglais est toujours employé dans le contexte de l'apprentissage.

Le théorème de Schapire sur la « force de l'apprentissage faible » prouve que H a une performance supérieure à celle de l'hypothèse qui aurait été apprise directement sur l'échantillon \mathcal{S} .

Une illustration géométrique du boosting selon cette technique de base est donnée dans les figures 11.12, 11.13 et 11.14.

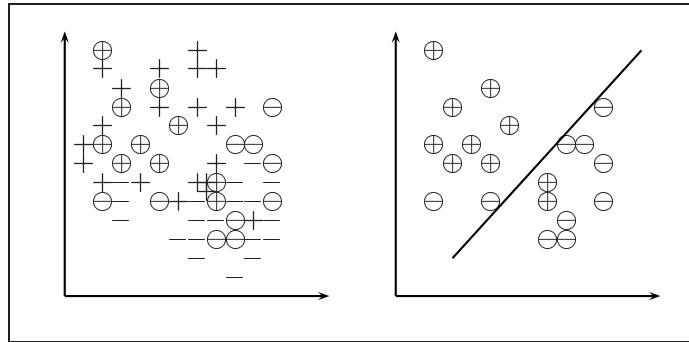


FIG. 11.12 – *À gauche : l'ensemble d'apprentissage \mathcal{S} et le sous-ensemble \mathcal{S}_1 (points entourés). À droite : l'ensemble \mathcal{S}_1 et la droite \mathcal{C}_1 apprise sur cet ensemble.*

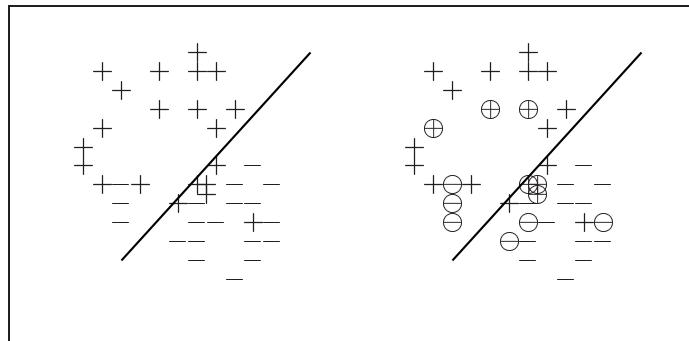


FIG. 11.13 – *À gauche : l'ensemble $\mathcal{S} - \mathcal{S}_1$ et la droite \mathcal{C}_1 apprise sur \mathcal{S}_1 . À droite : un ensemble \mathcal{S}_2 inclus dans $\mathcal{S} - \mathcal{S}_1$ parmi les plus informatifs pour \mathcal{C}_1 (points entourés).*

Idéalement, les trois ensembles d'exemples extraits de \mathcal{S} devraient le vider de tous ses exemples, ce qui revient à dire que la somme des valeurs m_1 , m_2 et m_3 doit approcher m . C'est la façon de tirer un profit maximal de \mathcal{S} . Mais on conçoit que ce réglage ne soit pas forcément facile à faire en pratique : si l'algorithme \mathcal{A} est performant sur \mathcal{S} , m_2 pourra être pris bien inférieur à m_1 , alors que la proportion pourrait être inverse si \mathcal{A} est seulement un peu meilleur qu'un tirage de classe au hasard. En général, on règle empiriquement les proportions des trois ensembles en faisant plusieurs essais, jusqu'à ce que tous les éléments de \mathcal{S} ou presque participent au processus.

On peut utiliser récursivement la méthode et procéder avec neuf sous-ensembles, vingt-sept sous-ensembles, etc. Mais la meilleure généralisation est de faire glisser la notion de fonction caractéristique (qui vaut 1 sur les points d'un sous-ensemble et 0 partout ailleurs) vers celle de distribution de probabilité sur les points de l'ensemble d'apprentissage. Cette technique sera également employée pour les fenêtres de Parzen (chapitre 14). C'est ce que réalise l'algorithme que nous présentons maintenant.

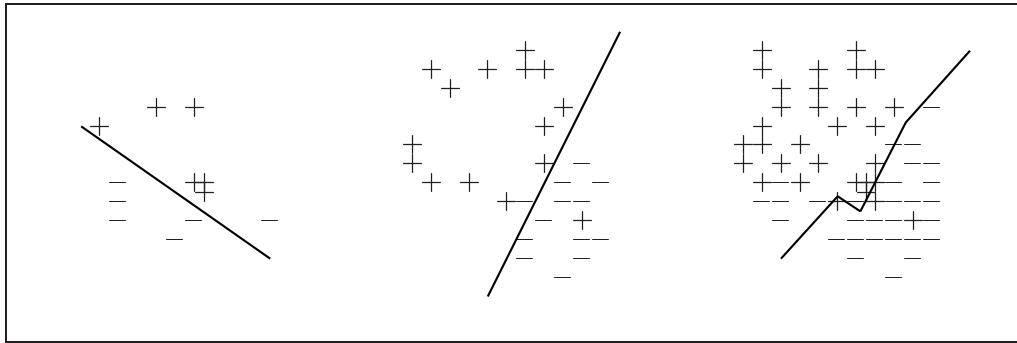


FIG. 11.14 – À gauche : l’ensemble S_2 et la droite séparatrice C_2 apprise sur cet ensemble. Au centre : l’ensemble $S_3 = S - S_1 - S_2$ et la droite séparatrice C_3 apprise sur cet ensemble. À droite : l’ensemble S et la combinaison des 3 droites séparatrices apprises sur cet ensemble.

11.3.3 Le boosting probabiliste et l’algorithme ADABoost

Trois idées fondamentales sont à la base des méthodes de boosting probabiliste :

1. L’utilisation d’un comité d’experts spécialisés que l’on fait voter pour atteindre une décision.
2. La pondération adaptative des votes par une technique de mise à jour multiplicative.
3. La modification de la distribution des exemples disponibles pour entraîner chaque expert, en surpondérant au fur et à mesure les exemples mal classés aux étapes précédentes.

L’algorithme le plus pratiqué s’appelle ADABoost (pour *adaptive boosting*). L’une des idées principales (voir l’algorithme 11.3) est de définir à chacune de ses étapes $1 \leq t \leq T$, une nouvelle distribution de probabilité *a priori* sur les exemples d’apprentissages en fonction des résultats de l’algorithme à l’étape précédente. Le poids à l’étape t d’un exemple $((x)_i, u_i)$ d’indice i est noté $D_t(i)$. Initialement, tous les exemples ont un poids identique, puis à chaque étape, les poids des exemples mal classés par l’apprenant sont augmentés, forçant ainsi l’apprenant à se concentrer sur les exemples difficiles de l’échantillon d’apprentissage.

À chaque étape t , l’apprenant cherche une hypothèse $h_t : \mathcal{X} \rightarrow \{-1, +1\}$ bonne pour la distribution D_t sur \mathcal{X} . La performance de l’apprenant est mesurée par l’erreur :

$$\varepsilon_t = p_{D_t}[h_t(\mathbf{x}_i) \neq u_i] = \sum_{i : h_t(\mathbf{x}_i) \neq u_i} D_t(i)$$

On note que l’erreur est mesurée en fonction de la distribution D_t sur laquelle l’apprenant est entraîné. En pratique, soit les poids des exemples sont effectivement modifiés, soit c’est la probabilité de tirage des exemples qui est modifiée et l’on utilise un tirage avec remise (*bootstrap*).

Chaque hypothèse h_t apprise est affectée d’un poids α_t mesurant l’importance qui sera donnée à cette hypothèse dans la combinaison finale. Ce poids est positif si $\varepsilon_t \leq 1/2$ (on suppose ici que les classes ‘+’ et ‘-’ sont équiprobales, et donc que l’erreur d’une décision aléatoire est de $1/2$). Plus l’erreur associée à l’hypothèse h_t est faible, plus celle-ci est dotée d’un coefficient α_t important¹⁶.

16. Le terme ADABoost vient du fait que contrairement aux algorithmes de boosting antérieurs, il n’est pas nécessaire de fournir la borne d’amélioration γ *a priori*. ADABoost s’adapte à l’erreur de chaque hypothèse faible.

L'examen des formules de mise à jour des poids des hypothèses dans l'algorithme 11.3 suggère que vers la fin de l'apprentissage, le poids des exemples difficiles à apprendre devient largement dominant. Si une hypothèse peut être trouvée qui soit performante sur ces exemples (c'est-à-dire avec $\varepsilon_t \approx 0$), elle sera alors dotée d'un coefficient α_t considérable. L'une des conséquences possibles est que les exemples bruités, sur lesquels finit par se concentrer l'algorithme, perturbent gravement l'apprentissage par boosting. C'est en effet ce qui est fréquemment observé.

Algorithme 11.3 ADABOOST dans le cas d'un apprentissage de concept

$\mathcal{S} = \{(\mathbf{x}_1, \mathbf{u}_1), \dots, (\mathbf{x}_m, \mathbf{u}_m)\}$, avec $\mathbf{u}_i \in \{+1, -1\}$, $i = 1, m$

pour tout $i=1,m$ faire

$p_0(\mathbf{x}_i) \leftarrow 1/m$

fin pour

$t \leftarrow 0$

tant que $t \leq T$ faire

Tirer un échantillon d'apprentissage \mathcal{S}_t dans \mathcal{S} selon les probabilités p_t

Apprendre une règle de classification h_t sur \mathcal{S}_t par l'algorithme \mathcal{A}

Soit ε_t l'erreur apparente de h_t sur \mathcal{S} . Calculer $\alpha_t \leftarrow \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$

pour $i = 1, m$ faire

$p_{t+1}(\mathbf{x}_i) \leftarrow \frac{p_t(\mathbf{x}_i)}{Z_t} e^{-\alpha_t}$ si $h_t(\mathbf{x}_i) = u_i$ (bien classé par h_t)

$p_{t+1}(\mathbf{x}_i) \leftarrow \frac{p_t(\mathbf{x}_i)}{Z_t} e^{+\alpha_t}$ si $h_t(\mathbf{x}_i) \neq u_i$ (mal classé par h_t).

(Z_t est une valeur de normalisation telle que $\sum_{i=1}^m p_t(\mathbf{x}_i) = 1$)

fin pour

$t \leftarrow t + 1$

fin tant que

Fournir en sortie l'hypothèse finale: $H(\mathbf{x}) = \text{sign}(\sum_{t=1}^T \alpha_t h_t(\mathbf{x}))$

À la fin de cet algorithme, chaque règle de classification h_t est pondérée par une valeur α_t calculée en cours de route. La classification d'un nouvel exemple (ou des points de \mathcal{S} pour obtenir l'erreur apparente) se fait en utilisant la règle :

$$H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^{t=T} \alpha_t h_t(\mathbf{x}) \right)$$

En un sens, on voit que le boosting construit l'hypothèse finale comme une série additive dans une base de fonctions, dont les éléments sont les hypothèses h_t . On retrouve là un thème fréquent dans les techniques d'apprentissage (par exemple les SVM, les méthodes d'approximation bayésiennes, etc.).

11.3.4 Les propriétés de l'algorithme ADABOOST

Commençons par analyser l'erreur en apprentissage de ADABOOST. Écrivons l'erreur ε_t de h_t comme : $\frac{1}{2} - \gamma_t$, où γ_t mesure l'amélioration apportée par l'hypothèse h_t par rapport à l'erreur de base 1/2. Freund et Shapire, [FS97], ont montré que l'erreur en apprentissage (la fraction

d'erreur sur l'échantillon d'apprentissage \mathcal{S}) de l'hypothèse finale H est bornée par :

$$\prod_t \left[2\sqrt{\varepsilon_t(1-\varepsilon_t)} \right] = \prod_t \sqrt{1 - 4\gamma_t^2} \leq \exp \left(-2 \sum_t \gamma_t^2 \right)$$

Ainsi, si chaque hypothèse faible est légèrement meilleure que le hasard, ($\gamma_t \geq \gamma > 0$), alors l'erreur en apprentissage diminue exponentiellement rapidement avec t .

L'erreur en généralisation de l'hypothèse finale H peut être bornée par une expression faisant intervenir l'erreur en apprentissage, le nombre d'exemple d'apprentissage m , la dimension de Vapnik-Chervonenkis $d_{\mathcal{H}}$ de l'espace d'hypothèse et T le nombre d'étapes de boosting ([FS97]) :

$$R_{R\uacute{e}el}(H) = R_{Emp}(H) + \mathcal{O} \left(\sqrt{\frac{T \cdot d_{\mathcal{H}}}{m}} \right)$$

où $R_{Emp}(H)$ dénote l'erreur empirique mesurée sur l'échantillon d'apprentissage.

Cette borne suggère que le boosting devrait tendre à surapprendre lorsque T devient grand, puisque le deuxième terme devient grand. Si cela arrive effectivement parfois, il a été observé empiriquement que souvent cela ne se produit pas. De fait, il apparaît même fréquemment que le risque réel tend à diminuer même longtemps après que le risque empirique soit devenu nul. En réponse à cette observation énigmatique, des chercheurs ont essayé d'établir un lien entre le boosting et les *méthodes à large marge* (voir le chapitre 9 sur les SVM).

Définissons la *marge* d'un exemple (\mathbf{x}, y) par :

$$\text{marge}(\mathbf{x}, y) = \frac{y \sum_{t=1}^T \alpha_t h_t(\mathbf{x})}{\sum_{t=1}^T \alpha_t}$$

Ce nombre est compris dans l'intervalle $[-1, +1]$ et est positif seulement si H classifie correctement l'exemple. La marge peut être interprétée comme une mesure de confiance dans la prédiction. Il a été prouvé que l'erreur en généralisation peut alors être bornée par :

$$R_{R\uacute{e}el}(H) \leq \hat{Pr}[\text{marge}(\mathbf{x}, y) \leq \theta] + \mathcal{O} \left(\sqrt{\frac{d_{\mathcal{H}}}{m\theta^2}} \right)$$

pour tout $\theta > 0$ avec forte probabilité.

On note que cette borne est maintenant indépendante de T , le nombre d'étapes de boosting. De plus, il a pu être montré que le boosting cherche effectivement à augmenter la marge avec les exemples puisqu'il se concentre sur les exemples difficiles à classer, c'est-à-dire sur les exemples dont la marge est la plus faible.

Schématiquement, ADABOOST et les méthodes SVM effectuent une recherche de classificateurs à large marge dans des espaces de grandes dimensions, ces classificateurs étant de plus des combinaisons linéaires, mais :

- les normes utilisées (L_2 pour les SVM et L_1 et L_∞ pour ADABOOST) sont différentes, donc les espaces explorés aussi
- l'optimisation sous contrainte est quadratique pour les SVM et linéaire pour le boosting
- la recherche est globale pour les SVM ce qui est rendu possible par l'astuce des fonctions noyau permettant de faire des calculs virtuels simples dans des espaces de très grande dimension, tandis que le boosting effectue une recherche locale gloutonne (une coordonnée $h(\mathbf{x})$ à la fois, cette coordonnée devant avoir une corrélation non négligeable (meilleure que le hasard) avec l'étiquette y).

Si des liens ont ainsi pu être établis de manière prometteuse entre le boosting et les méthodes à large marge, il reste encore à les investiguer de manière plus complète et il y a là encore de beaux sujets de recherche en perspective (voir par exemple [FS99] pour des références).

11.3.5 L'utilisation du boosting

Le boosting, et particulièrement l'algorithme ADABOOST, a été employé avec succès avec de nombreux algorithmes d'apprentissage « faibles » (par exemple C4.5 : un système d'apprentissage d'arbre de décision ([Qui93]) ou RIPPER : un système d'apprentissage de règles) et sur des domaines d'application variés. En général, l'utilisation du boosting a pour résultat d'améliorer souvent sensiblement les performances en apprentissage.

Les avantages du boosting et de ADABOOST en particulier sont qu'il s'agit d'une méthode facile à programmer et aisée d'emploi. Elle ne nécessite pas de connaissance *a priori* sur l'algorithme d'apprentissage « faible » utilisé, et elle peut s'appliquer de fait à n'importe quel algorithme d'apprentissage faible. Les seuls paramètres à régler sont la taille de l'ensemble d'apprentissage m et le nombre total d'étapes T , qui peuvent être fixés par l'utilisation d'un ensemble de validation (voir le chapitre 3). De plus, des garanties théoriques sur l'erreur en généralisation permettent de contrôler l'apprentissage. Une autre propriété intéressante du boosting est qu'il tend à détecter les exemples aberrants (*outliers*) puisqu'il leur donne un poids exponentiellement grand en cours d'apprentissage. Cependant, la contrepartie de ce phénomène est que le boosting est sensible au bruit et ses performances peuvent être grandement affectées lorsque de nombreux exemples sont bruités. Récemment des algorithmes ont été proposés pour traiter ce problème (comme *Gentle AdaBoost* [HTF01] ou *BrownBoost* [Fre99]).

Il est à noter que l'adaptation aux problèmes multiclasses n'est pas immédiate, mais elle a cependant fait l'objet d'études menant aussi à des algorithmes efficaces. De même qu'il existe des extensions à la régression.

11.3.6 Boosting et théorie PAC

Les premiers travaux sur le boosting sont issus d'une question posée par Valiant et Kearns [KV88] dans le cadre de l'apprentissage *PAC*. Les algorithmes *PAC* au sens *fort* sont définis ainsi :

- pour toute distribution de probabilité $\mathcal{D}_{\mathcal{X}} \times \mathcal{D}_{\mathcal{U}}$ sur l'espace des exemples (\mathbf{x}, \mathbf{u}) ,
- $\forall \varepsilon > 0, \delta > 0$,
- étant donné un nombre polynomial (fonction de $1/\varepsilon$ et de $1/\delta$) d'exemples i.i.d. suivant $\mathcal{D}_{\mathcal{X}} \times \mathcal{D}_{\mathcal{U}}$,
- l'algorithme trouve une hypothèse d'erreur $\leq \varepsilon$ avec une probabilité $\geq 1 - \delta$.

Les algorithmes d'apprentissage dits *faibles* ont une définition analogue, mais on leur demande seulement de trouver une hypothèse d'erreur $\varepsilon \geq \frac{1}{2} - \gamma$, avec γ strictement positif, donc éventuellement juste un peu meilleure que le hasard, en supposant une tâche de classification binaire avec la même proportion d'exemples positifs et négatifs.

La question posée : est-ce qu'il est possible d'utiliser un algorithme faible pour obtenir un apprentissage de type fort ?

Shapire ([FS99]) a prouvé que la réponse à cette question est positive et a conçu le premier algorithme de boosting par sous-ensembles. Freund [Fre99] a ensuite produit un algorithme beaucoup plus efficace, également optimal, mais difficile à appliquer. En 1995, Freund et Shapire [FS97] ont proposé l'algorithme ADABOOST, efficace et pratique, qui est maintenant la technique la plus employée pour améliorer les performances de n'importe quel algorithme d'apprentissage supervisé.

Dans le même temps, d'autres chercheurs ont analysé comment il est possible d'identifier les bons experts au sein d'une grande collection d'experts ou bien les bons attributs quand on a un grand nombre d'attributs (ces deux problèmes sont reliés). Les algorithmes développés, tels que *Winnow* [LW94] (voir au chapitre 15) ont révélé l'intérêt de la mise à jour multiplicative des pondération d'experts.

11.3.7 Le « bagging »

Le *bagging* est une méthode qui, comme le boosting, combine des hypothèses pour obtenir une hypothèse finale. Cependant la méthode est plus simple et généralement moins performante. L'idée de base est d'entraîner un algorithme d'apprentissage (arbre de décision, réseau connexionniste, etc.) sur plusieurs bases d'apprentissage obtenues par tirage avec remise¹⁷ de m' (avec $m' < m$) exemples d'apprentissage dans l'échantillon d'apprentissage \mathcal{S} . Pour chaque tirage b (pour *bag*), une hypothèse h_b est obtenue. L'hypothèse finale est simplement la moyenne des hypothèses obtenues sur B tirages au total :

$$H(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^B h_b(\mathbf{x})$$

L'une des justifications de cette méthode est que si les hypothèses h_b calculées pour chaque tirage b ont une variance importante (donc sont sensibles à l'échantillon des m' exemples d'apprentissage), alors leur moyenne H aura une variance réduite.

Notes historiques et sources bibliographiques

Un certain nombre de travaux préliminaires sur la reconnaissance des formes par des méthodes hiérarchiques ont été compilés dans des articles bibliographiques, par exemple [SL91], avant la parution en 1984 des travaux décisifs dans « CART », le livre vraiment fondateur des techniques des arbres de décision et de régression ([BFOS84]). Cet ouvrage ne développe pas seulement l'algorithme d'apprentissage de base, mais explique la validité statistique de l'élagage et donne des exemples sur des attributs binaires et numériques. La relève est ensuite principalement prise par R. Quinlan, qui développe les algorithmes ID3 et C4.5 ([Qui93], [QR89]) et applique la méthode à des données numériques et symboliques variées. Un aspect original, le développement incrémental des arbres de décision, a été proposé par P. Utgoff ([Utg89]).

En lisant les ouvrages récents sur la fouille de données, par exemple [HK01], on constate l'importance pratique de ces méthodes, encore une fois pratiquement les seules à savoir traiter de manière homogène les exemples décrits par (presque) tous les types d'attributs.

Le livre récent de Zighed et Rakotomalala [ZR00] dresse un panorama complet sur les arbres de décision et analyse leur extension à des *graphes*, tout en présentant une grande variété d'applications, en particulier à la classification non supervisée et à la régression. Il donne aussi une remarquable bibliographie.

On trouvera des exposés pédagogiques sur les arbres de décision dans de nombreux livres. Les chapitres sur le sujet dans [WK91] et [Mit97] sont particulièrement didactiques. On pourra consulter une bonne bibliographie dans [Mit97] et [DHS01].

Le matériel présenté ici pour les arbres de décision a été en particulier inspiré par le texte de O. Gascuel dans [Nic93]. On reprend dans ce chapitre le critère de sélection d'un attribut par l'entropie proposé par Quinlan dans la méthode ID3 et la technique d'élagage de CART.

17. Méthode de tirage que l'on appelle *bootstrap*.

Le critère de Lerman(11.6) a été aimablement reformulé par son auteur pour s'adapter à nos notations.

Le logiciel OC1 ([MKS94]) et les données de Fisher (avec quatre attributs) sont disponibles au public, comme beaucoup d'autres jeux de données, à partir du site Internet :

<http://www.ai.univie.ac.at/oefai/ml/ml-ressources.html>

L'histoire du boosting est évoquée au paragraphe 11.3.6. De remarquables développements théoriques ont été effectués sur ces méthodes en particulier sur la capacité de généralisation et les liens avec les *SVM* (chapitre 9). Le mot anglais *arcing* (de *adaptive reweighting and combining*) est employé pour désigner toutes les méthodes qui sélectionnent ou répondent aux données pour améliorer la classification. Les deux méthodes de boosting que nous avons vues en font partie. Le *bagging* (de *bootstrap aggregation*) en fait partie aussi (voir [HTF01] et le chapitre 3).

Résumé

Les *arbres de décision* permettent de classer des objets ayant des attributs de nature discrète. Ils sont construits récursivement par spécialisation progressive sur l'espace des exemples. La classe d'un objet est ainsi prédite par la classe de la région calculée par l'arbre de décision dans laquelle il se trouve.

- À chaque nœud correspond un test à n valeurs (souvent binaire) sur un attribut.
- On classe un objet en partant de la racine et en suivant le chemin dans les nœuds donné par la réponse aux tests, jusqu'à une feuille (à laquelle est affectée une classe).
- L'apprentissage se fait récursivement, en choisissant pour racine de l'arbre l'attribut le plus corrélé avec la distribution en classes, ce qui sépare les exemples en n parties sur lesquelles on recommence ce choix.
- On arrête l'apprentissage lorsque les feuilles de l'arbre sont suffisamment « pures » ou qu'aucun test n'est plus disponible.
- une phase d'élagage est ensuite nécessaire pour réduire l'arbre et diminuer sa probabilité d'erreur de classification.
- De par leur facilité d'utilisation et la relative transparence des hypothèses produites, les arbres de décision sont très largement employés.

Le *boosting* est une technique d'apprentissage qui vise à rendre plus performant un système d'apprentissage « faible ». Pour ce faire, le système d'apprentissage est entraîné successivement sur des échantillons d'apprentissage surpondérant les exemples difficiles à apprendre. À chaque fois, une hypothèse h_t est produite, et l'hypothèse finale est une combinaison linéaire de ces hypothèses pondérées par des coefficients liés à leur performance. Le boosting est d'un emploi très large et fait l'objet de nombreux travaux et applications.

Chapitre 12

L'apprentissage de réseaux bayésiens

On présente dans ce chapitre un cas particulier de modèles permettant d'exprimer des relations probabilistes entre des ensembles de faits. Ces relations diffèrent des relations logiques en ce qu'elles n'autorisent pas un raisonnement implicatif, mais conditionnel. Deux faits peuvent en effet être en relation causale sans que l'un implique l'autre. Dans la base de données d'une compagnie d'assurances, il est par exemple possible que, pour une majorité des entrées dans une certaine ville, les items « contravention de stationnement » soient vrais quand un autre item comme « le conducteur aime les légumes » soient corrélés. Il serait trop rapide et peu fructueux d'en conclure que le second fait implique statistiquement le premier. Une analyse en probabilité conditionnelle des faits pourrait en effet révéler que la majorité des contraventions sont collées le samedi, jour du marché. Il existe bien une « cause » commune (ou en tout cas une condition commune de forte probabilité), mais la logique n'a pas réellement à intervenir dans cette affaire. L'ensemble des faits et des probabilités conditionnelles d'un système de raisonnement de ce type peut s'organiser en graphe, sous certaines conditions d'indépendance probabiliste. On peut alors raisonner, c'est-à-dire calculer la probabilité conditionnelle de n'importe quel ensemble de faits connaissant n'importe quel autre ensemble. C'est une technique puissante pour exploiter utilement les bases de données. Cependant, un système de ce type ne peut être complètement utile que si un programme est capable d'extraire automatiquement les faits significatifs et le réseau de leurs relations conditionnelles. Une base de données comporte couramment des dizaines de milliers d'entrées ventilées sur des centaines de faits VRAI ou FAUX. Aucun expert ne peut extraire seul une structure de dépendance probabiliste d'une telle quantité de données. C'est ici qu'intervient l'apprentissage artificiel...

CE MATIN, il y a un cygne à bec jaune sur ma pelouse. Deux possibilités: soit il s'est échappé du zoo voisin, soit c'est un oiseau migrateur. Pour le savoir, je me dirige vers le poissonnier du coin et j'observe que le prix de saumon norvégien a augmenté. J'en déduis que le cygne est sauvage.

Ce genre de « raisonnement » possède deux caractéristiques un peu surprises. La première est que la décision prise n'est pas forcément la bonne : le cygne peut très bien s'être échappé du zoo et je peux aussi me tromper à propos du cours du saumon. Surtout, il n'y a pas d'implication (de relation de cause à effet) entre le fait d'avoir un cygne sur ma pelouse et celui d'avoir à payer le saumon plus cher. Cependant, constater le second change ma confiance dans l'origine du premier. Pourquoi ? S'il fait très froid dans le Nord de l'Europe, deux phénomènes en apparence indépendants ont une probabilité forte d'arriver : la migration jusqu'à nos latitudes d'une espèce qui est en général plus septentrionale et l'augmentation du cours du poisson récolté sur place, plus difficile à pêcher et à traiter.

On peut isoler quatre faits dans cet univers, associés à une probabilité d'être *VRAI* : il y a un cygne sur ma pelouse, le zoo est mal surveillé, le saumon a augmenté, l'hiver est froid dans le Nord. Les données de départ sont les suivantes : d'abord, il y a un cygne sur ma pelouse avec une très forte probabilité (je suis expert ornithologue et peu sujet à des hallucinations). Ensuite, le prix du saumon norvégien a augmenté, avec une bonne probabilité. D'autre part, je connais un certain nombre de relations comme :

- Les cygnes à bec jaune migrent parfois jusqu'ici quand il fait froid dans le Nord.
- Les animaux peuvent s'échapper du zoo si celui-ci a eu un problème technique.
- Le prix de saumon augmente quand les conditions de pêche sont mauvaises dans le Nord de l'Europe.

Je dois pour chaque relation donner la valeur de deux probabilités conditionnelles : celle que ce cygne sauvage soit sur ma pelouse sachant qu'il fait particulièrement froid dans le Nord et celle que ce cygne sauvage soit sur ma pelouse sachant qu'il ne fait pas particulièrement froid dans le Nord (la somme des deux ne vaut pas forcément 1). Et de même pour les deux autres relations.

Un cygne est sur ma pelouse soit parce c'est un évadé du zoo, soit parce que c'est un migrateur. Les deux causes sont, disons, *a priori* plausibles au même degré. Mais observer l'augmentation du prix du saumon m'indique qu'il est probable qu'il y ait des mauvaises conditions de pêche en Norvège (dans mon ensemble de faits, c'est le seul qui puisse intervenir). Donc, l'hypothèse que le cygne est migrateur devient plus vraisemblable que celle de la défaillance technique au zoo.

Dit autrement : s'il fait très froid dans le Nord de l'Europe, deux phénomènes en apparence indépendants ont une probabilité forte d'arriver ensemble : la migration jusqu'à nos latitudes d'une espèce qui est en général plus septentrionale et l'augmentation du cours du poisson récolté sur place.

Supposons maintenant que j'observe un certain nombre de fois les quatre faits cités ci-dessus. Puis-je *apprendre* à raisonner ainsi en me trompant le moins possible dans la conclusion ? C'est en effet le sujet de ce chapitre que de montrer comment raisonner dans un « réseau » de probabilités conditionnelles et surtout comment apprendre à le construire à partir d'observations conjointes des faits de base.

12.1 Les réseaux d'inférence bayésiens

Les *réseaux d'inférence bayésiens* sont des modèles qui permettent de représenter des situations de raisonnement probabiliste à partir de connaissances incertaines. Ils sont une représentation

tion efficace pour les calculs d'une distribution de probabilités. Plus précisément, les réseaux bayésiens conjuguent deux aspects (voir la figure 12.1) :

- Une **partie qualitative** exprimant des indépendances conditionnelles entre variables et des liens de causalité. Cela se fait grâce à un graphe orienté acyclique¹ dont les nœuds correspondent à des variables aléatoires (dont nous supposerons qu'elles ne peuvent prendre qu'un ensemble fini de valeurs, et même souvent les seules valeurs *VRAI* et *FAUX*).
- Une **partie quantitative** constituée des tables de probabilités conditionnelles de chaque variable étant donnés ses parents dans le graphe.

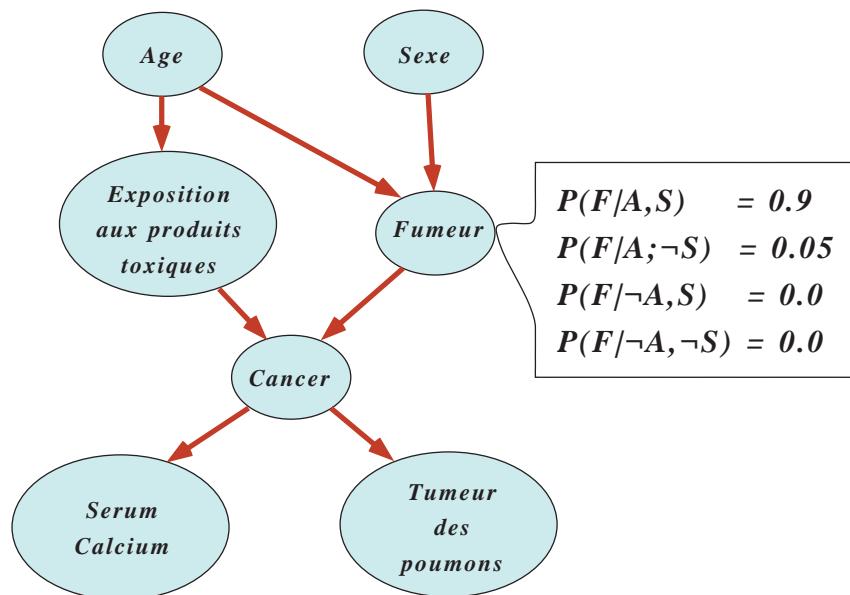


FIG. 12.1 – Les réseaux bayésiens sont une représentation compacte et efficace pour les calculs d'une distribution de probabilités grâce à l'exploitation des indépendances conditionnelles entre variables. À chaque nœud est associée une table de probabilités conditionnelles.

Prises ensemble ces deux parties définissent une distribution de probabilité unique sous forme factorisée. Par exemple, dans le cas de la figure 12.1 :

$$\begin{aligned} P(A, S, E, F, C, Se, T) = \\ p(A) \cdot P(S) \cdot P(E|A) \cdot P(F|A, S) \cdot P(C|E, F) \cdot P(Se|C) \cdot P(T|C) \end{aligned}$$

Cette factorisation prend en compte les indépendances conditionnelles exprimées dans le graphe. Ainsi par exemple, *Serum Calcium* et *Tumeur des poumons* sont deux variables dépendantes, mais elles deviennent indépendantes si l'on connaît la valeur de la variable *Cancer*.

Si l'on ne tenait pas compte de ces indépendances, il faudrait écrire la distribution jointe de probabilités comme :

$$\begin{aligned} P(A, S, E, F, C, Se, T) = \\ p(A) \cdot P(S|A) \cdot P(E|A, S) \cdot P(F|A, S, E) \cdot \\ P(C|A, S, E, F) \cdot P(Se|A, S, E, F, C) \cdot P(T|A, S, E, F, C, Se) \end{aligned}$$

1. On dit souvent : un *DAG*, de l'anglais *directed acyclic graph*.

La distribution de probabilités totale nécessite donc ici la connaissance de 16 paramètres au lieu de $2^{15} = 32768$ si on ne tenait pas compte des indépendances conditionnelles encodées dans le graphe. Plus grand est le nombre de variables et plus l'économie est potentiellement considérable. Nous verrons également que les inférences, et l'apprentissage en seront facilités d'autant.

Chaque fait est représenté graphiquement par un nœud et les relations directes entre nœuds sont des arcs orientés entre les nœuds. L'exemple de l'introduction comporte donc quatre nœuds et trois arcs. Complètement défini, il représenterait de manière condensée toute l'information sur les dépendances entre les faits.

Nous allons d'abord définir un réseau d'inférence bayésien en y introduisant toutes les valeurs de probabilités nécessaires ; nous verrons ensuite comment un tel modèle permet de raisonner. Enfin, nous donnerons quelques techniques pour apprendre ces modèles à partir d'exemples.

12.1.1 Définitions et notations

Un *réseau d'inférence bayésien*, ou *réseau bayésien*, est un système de raisonnement probabiliste construit sur un graphe orienté sans cycle. Nous emploierons donc dans ce chapitre le vocabulaire classique de la théorie des graphes : par exemple, un nœud F sera dit *descendant* d'un nœud A s'il existe un *chemin* (une suite d'arcs, ici orientés) entre F et A . Les termes de *parent*, *descendant direct* ou *fils*, *descendant*, *non-descendant* et *ancêtre* seront utilisés (des exemples sont donnés à la figure 12.2). Chaque nœud d'un réseau bayésien porte une étiquette

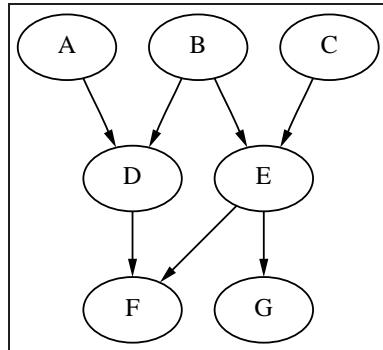


FIG. 12.2 – *Un graphe orienté sans cycle. Les parents de F sont D et E. Les ancêtres de F sont D, E, A, B et C. Les fils de B sont D et E. Les descendants de B sont D, E, F et G. Les non-descendants de A sont B, C, E et G.*

qui est un des attributs du problème. Ces attributs sont binaires, pouvant prendre (avec une certaine probabilité) la valeur *VRAI* ou *FAUX*, ce qui signifie qu'une variable aléatoire est associée à chaque attribut. Comme à chaque nœud est associée un attribut, donc une variable aléatoire différente, nous pouvons confondre par la suite un nœud, un attribut et la variable aléatoire associée.

Nous notons la probabilité que la variable X soit *VRAI* par $P(X = \text{VRAI})$, ou en raccourci $P(X)$. On a : $P(X = \text{FAUX}) = 1 - P(X = \text{VRAI})$, ce que nous notons : $P(\neg X) = 1 - P(X)$.

12.1.2 La d-séparation

Les indépendances conditionnelles encodées par le graphe sont calculables grâce à un critère formel de théorie des graphes que l'on appelle la *d-séparation*. Ce critère permet le calcul des indépendances conditionnelles en temps polynomial en fonction du nombre de variables.

Ainsi par exemple, chaque noeud X soit indépendant de tout autre noeud qui n'est pas son descendant, indépendant *sachant* les parents de X . Ou encore:

Dans un réseau bayésien, tout noeud est conditionnellement indépendant de ses non-descendants, sachant ses parents.

En termes plus formels, notons $\mathcal{A}(V)$ n'importe quel ensemble de noeuds qui ne sont *pas* des descendants de V et $\mathcal{P}(V)$ l'ensemble des parents de V . Ceci s'écrit :

$$P(V|\mathcal{A}(V), \mathcal{P}(V)) = P(V|\mathcal{P}(V)) \quad (12.1)$$

Autrement dit, l'ensemble des valeurs $P(V|\mathcal{P}(V))$, avec V parcourant l'ensemble des noeuds du graphe, suffit à déterminer complètement l'ensemble de toutes les probabilités conditionnelles d'un réseau bayésien.

Compte tenu de la structure particulière du graphe, on peut démontrer [Fre98] que la condition 12.1 peut se réécrire sous la forme suivante :

Théorème 12.1

Soit $\mathcal{V} = \{v_1, \dots, v_k\}$ l'ensemble des noeuds du graphe. On a :

$$P(v_1, \dots, v_k) = \prod_{i=1}^k P(v_i|\mathcal{P}(v_i)) \quad (12.2)$$

Exemple 5

Nous adaptons de [Nil98] l'exemple dont le graphe est donné à la figure 12.3.

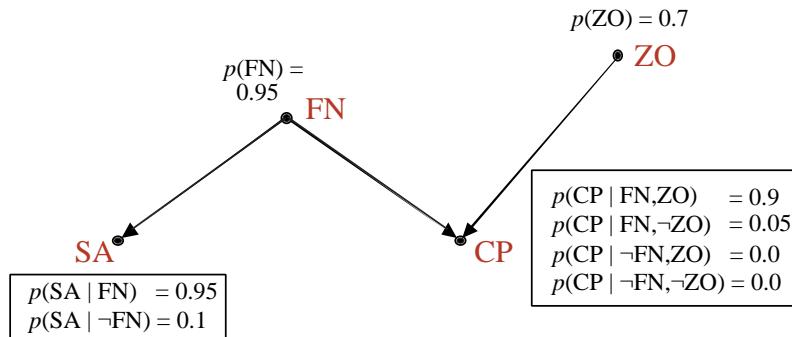


FIG. 12.3 – Le graphe du réseau bayésien qui exprime « Il y a un cygne à bec jaune sur ma pelouse ».

Ce réseau reprend le problème de l'introduction, avec les notations suivantes :

- FN: Il fait **Froid** en **Norvège**;
- ZO: La cage des cygnes est ouverte au **ZOo**;
- CP: Il y a un **Cygne sauvage** sur ma **Pelouse**;
- SA: Le prix de **Saumon** a **Augmenté**;

FN et ZO ont une influence directe sur CP puisque ce sont donc ses noeuds parents. De même, FN a une influence directe sur SA .

La spécification complète d'un réseau bayésien doit affecter à chaque variable toutes les probabilités conditionnelles significatives (celles de l'équation 12.2). Compte tenu des dépendances de notre exemple, il faut donc d'abord spécifier les valeurs $P(FN)$ et $P(ZO)$ qui sont non

conditionnelles, puisque les nœuds correspondants sont tous les deux sans ancêtres. Notons que les variables FN et ZO sont indépendantes pour la même raison. Il nous faut, pour le nœud SA , donner $P(SA | FN)$ et $P(SA | \neg FN)$ et, pour le nœud CP , donner quatre valeurs : $P(CP | FN, ZO), P(CP | FN, \neg ZO), P(CP | \neg FN, ZO)$ et $P(CP | \neg FN, \neg ZO)$.

Notre réseau sera donc complet si nous lui ajoutons par exemple le tableau :

$P(FN)$	0.95
$P(ZO)$	0.7
$P(SA FN)$	0.95
$P(SA \neg FN)$	0.1
$P(CP FN, ZO)$	0.9
$P(CP FN, \neg ZO)$	0.05
$P(CP \neg FN, ZO)$	0
$P(CP \neg FN, \neg ZO)$	0

Ces valeurs sont données arbitrairement pour spécifier notre exemple. Par exemple, $P(SA | FN) = 0.95$ signifie que nous posons qu'il y a 95 % de chances que le prix du saumon augmente s'il fait froid en Norvège et $P(SA | \neg FN) = 0.1$ qu'il y a 10 % de chances qu'il augmente sans qu'il fasse particulièrement froid dans ce pays. Ces valeurs peuvent être obtenues par apprentissage, comme nous le verrons au paragraphe 12.3.

12.1.3 Définition formelle d'un réseau bayésien

Compte tenu de ce qui a été dit, nous pouvons maintenant donner une définition complète d'un réseau bayésien.

Définition 12.1

Un réseau bayésien est un couple (G, P) , avec :

- G est un graphe orienté sans cycle
- À chaque nœud de G est associée une variable aléatoire et une seule.
- Soit $\{v_1, \dots, v_n\}$ l'ensemble de ces variables aléatoires. La propriété suivante découle de la structure de graphe sans cycle :

$$P(v_1, \dots, v_k) = \prod_{i=1}^k P(v_i | \mathcal{P}(v_i)) \quad (12.3)$$

avec $\mathcal{P}(v_i)$ l'ensemble des variables associées aux parents du nœud associé à v_i .

Un réseau bayésien est donc complètement spécifié quand son graphe a été décrit et quand, pour chaque nœud de ce graphe, les probabilités conditionnelles de ce nœud sachant chacun de ces parents sont données.

12.2 Les inférences dans les réseaux bayésiens

Un réseau bayésien est donc un graphe causal auquel on a associé une représentation probabiliste sous-jacente. La correspondance qui existe entre la structure graphique et la structure probabiliste associée² permet de ramener l'ensemble des problèmes d'inférence à des problèmes de théorie des graphes, qui restent cependant assez complexes. Voyons cela de plus près.

2. C'est pourquoi on parle aussi fréquemment de *modèles graphiques* pour désigner les réseaux bayésiens.

Une fois qu'un réseau bayésien a été construit pour rendre compte d'un domaine, on cherche souvent à l'utiliser pour déterminer des probabilités correspondant à certains événements, certaines questions, certaines dépendances. En général, ces probabilités ne sont pas stockées dans le réseau et il faut donc les calculer. Comme un réseau bayésien encode la distribution de probabilité jointe pour l'ensemble des variables du domaine, il permet en principe de calculer n'importe quelle probabilité d'intérêt. Les règles du calcul probabiliste sont utilisées pour cela. Dans la pratique, on rencontre cependant deux difficultés : d'une part, les calculs ne sont vraiment possibles que pour des variables à valeur discrète, d'autre part, il faut savoir tenir compte d'indépendances conditionnelles dans le réseau pour maîtriser la complexité des calculs. Plusieurs méthodes ont été mises au point pour cela.

12.2.1 Schémas d'inférence

Nous cherchons maintenant à calculer d'autres probabilités conditionnelles pour exprimer l'influence des variables les unes sur les autres. Par exemple, comment le réseau donné en exemple peut-il répondre aux questions :

- Quelle est la probabilité pour qu'il y ait un cygne sur ma pelouse sachant que le zoo a laissé une cage ouverte ?
- Quelle est la probabilité que le zoo soit en état normal sachant qu'il ne fait pas particulièrement froid en Norvège ?
- Quelle est la probabilité que le zoo soit en état normal sachant que le saumon n'a pas augmenté et qu'il ne fait pas particulièrement froid en Norvège ?

En termes formels, il s'agit de calculer les probabilités conditionnelles $P(CP|ZO)$, $P(\neg ZO|\neg CP)$ et $P(\neg ZO | \neg FN, \neg CP)$. Le premier cas s'appelle une *inférence causale* ou *descendante* : en effet, le noeud ZO est un ancêtre du noeud CP . On peut donc considérer ZO comme une *cause*³ de CP . Mais cela ne veut pas dire qu'il n'y ait pas de dépendance inverse. Le second calcul s'appelle une *inférence ascendante* ou un *diagnostic* : il s'agit de comprendre une variable par ses conséquences. Le troisième cas est une combinaison des deux premiers, appelée une *explication*.

D'une manière générale, le calcul de ces probabilités conditionnelles se base sur la formule de Bayes :

$$P(X, Y) = P(X|Y)P(Y) = P(Y|X)P(X)$$

et sur la règle du *chaînage* des probabilités conditionnelles :

$$P(X_1, \dots, X_n) = P(X_n|X_{n-1}, \dots, X_1)P(X_{n-1} | X_{n-2}, \dots, X_1) \dots P(X_2|X_1)P(X_1)$$

$$P(X_1, \dots, X_n) = \prod_{i=1}^n P(X_i|X_{i-1}, \dots, X_1)$$

Par exemple, pour quatre variables :

$$P(X_1, X_2, X_3, X_4) = P(X_4|X_3, X_2, X_1)P(X_3|X_2, X_1)P(X_2|X_1)P(X_1)$$

Exemple 6 (Un calcul d'inférence causale)

Pour calculer $P(CP|ZO)$, il faut faire intervenir l'autre parent de CP :

$$P(CP|ZO) = P(CP, FN|ZO) + P(CP, \neg FN|ZO)$$

3. Le mot de *cause* est à considérer avec prudence : il ne signifie pas ici implication logique.

puis « conditionner » ce parent par rapport à ZO :

$$P(CP|ZO) = P(CP|FN, ZO)P(FN|ZO) + P(CP|\neg FN, ZO)P(\neg FN|ZO)$$

Nous savons, puisque FN n'a pas de parent, que $P(FN|ZO) = P(FN)$ et que de même $P(\neg FN|ZO) = P(\neg FN)$.

D'où :

$$\begin{aligned} P(CP|ZO) &= P(CP|FN, ZO)P(FN) + P(CP|\neg FN, ZO)P(\neg FN) \\ &= 0.9 \times 0.95 + 0. \times (1 - 0.95) = 0.855 \end{aligned}$$

La généralisation du calcul fait sur cet exemple est facile à imaginer. On la trouvera développée formellement dans [Nil98] et [BN99].

Exemple 7 (Un calcul de diagnostic)

Ici, il faut renverser l'ordre des variables en utilisant la règle de Bayes :

$$P(\neg ZO|\neg M) = \frac{P(\neg M|\neg ZO)P(\neg ZO)}{P(\neg M)}$$

Nous pouvons maintenant calculer $P(\neg M, \neg ZO)$ par inférence causale :

$$\begin{aligned} P(\neg CP, \neg ZO) &= P(\neg CP, FN|\neg ZO) + P(\neg CP, \neg FN|\neg ZO) \\ &= P(\neg CP|FN, \neg ZO)P(FN, \neg ZO) \\ &\quad + P(\neg CP|\neg FN, \neg ZO)P(\neg FN, \neg ZO) \\ &= P(\neg CP|FN, \neg ZO)P(FN) + P(\neg CP|\neg FN, \neg ZO)P(\neg FN) \\ &= (1 - 0.05) \times 0.95 + (1.) \times 0.05 = 0.9525 \end{aligned}$$

D'où :

$$P(\neg ZO, \neg CP) = \frac{0.9525 \times P(\neg ZO)}{P(\neg CP)} = \frac{0.9525 \times 0.3}{P(\neg CP)} = \frac{0.28575}{P(\neg CP)} \quad (12.4)$$

Nous ne connaissons pas $P(\neg CP)$, mais nous contournons la difficulté en le traitant comme un facteur de normalisation en calculant, sans donner encore une fois le détail :

$$P(ZO|\neg CP) = \frac{P(\neg CP | ZO)P(ZO)}{P(\neg CP)} = \frac{0.0595 \times 0.7}{P(\neg CP)} = \frac{0.03665}{P(\neg CP)} \quad (12.5)$$

Comme $P(\neg ZO|\neg M) + P(ZO|\neg CP) = 1$, on déduit des équations 12.4 et 12.5 :

$$P(\neg ZO|\neg M) = 0.88632$$

La généralisation de ce calcul à tout réseau bayésien est également facile.

Exemple 8 (Un calcul d'explication)

Ici, le calcul se fait en employant à la fois la règle de Bayes et la règle de chaînage des probabilités conditionnelles.

$$\begin{aligned}
 P(\neg ZO | \neg FN, \neg CP) &= \frac{P(\neg CP, \neg FN, \neg ZO)P(ZO)}{P(\neg FN, \neg CP)} && \text{Règle de Bayes} \\
 &= \frac{P(\neg CP | \neg FN, \neg ZO)P(\neg FN | \neg ZO)P(\neg ZO)}{P(\neg FN, \neg CP)} && \text{Définition} \\
 &&& \text{des probabilités} \\
 &&& \text{conditionnelles} \\
 &= \frac{P(\neg CP | \neg FN, \neg ZO)P(\neg FN)P(\neg ZO)}{P(\neg FN, \neg CP)} && \text{FN et } ZO \\
 &&& \text{indépendants}
 \end{aligned}$$

Tous les termes de cette expression sont définies par le réseau, sauf $P(\neg FN, \neg CP)$, que l'on peut calculer par diagnostic. On trouve finalement :

$$P(\neg ZO | \neg FN, \neg CP) = 0.03$$

On peut remarquer que cette valeur est inférieure à $P(\neg ZO, \neg CP)$, ce qui signifie que savoir en plus que le prix du saumon n'a pas augmenté réduit la probabilité que le zoo soit en état normal sachant qu'il ne fait pas spécialement froid en Norvège. Voilà un résultat pour lequel l'intuition n'est pas très utile.

La généralisation de ce calcul à tout réseau bayésien est encore possible, mais l'organisation des calculs demande évidemment un algorithme plus complexe.

Complexité des calculs

Comme on l'a entrevu ci-dessus, il est donc possible en organisant correctement les calculs de dériver (à partir de la structure et des probabilités données au départ) toutes les probabilités conditionnelles du type $P(\mathcal{V}|\mathcal{W})$, où \mathcal{V} et \mathcal{W} sont des ensembles de noeuds.

Malheureusement, il a été démontré que ce calcul est NP-complet, c'est-à-dire qu'il n'existe vraisemblablement pas, si \mathcal{V} et \mathcal{W} sont quelconques, d'algorithme dont le temps de calcul soit polynomial en fonction du nombre total de noeuds du réseau. Il faut donc chercher à réduire ce temps de calcul en tenant compte le mieux possible de la structure, ce qui peut se faire de deux manières. La première technique consiste à trouver une relation entre la géométrie du graphe du réseau et l'*indépendance conditionnelle* de sous-ensembles de ses noeuds en étudiant la *d-séparation* des noeuds du graphe. La seconde consiste à contraindre la structure pour que les calculs se développent facilement (par exemple dans les graphes particuliers que sont les *polyarbres*).

Encore n'avons-nous parlé là que de réseaux sans boucles. Dans le cas de réseaux à boucles, c'est-à-dire dans lesquels il peut exister plusieurs chemins entre deux noeuds, il faut avoir recours à d'autres techniques, principalement :

- Les **méthodes de conditionnement** dans lesquelles on cherche à étendre les propriétés d'indépendance conditionnelles dans le graphe en cherchant des sous-ensembles de variables séparant d'autres sous-ensembles de variables.
- Les **méthodes de regroupement** (*méthode des arbres de jonction*) . Elles consistent à se ramener à un réseau sans boucle en créant des noeuds plus complexes qui représentent plusieurs noeuds du graphe original. Nous ne parlerons pas davantage de ces méthodes ici (voir sur ce sujet par exemple [BN99]).
- Les **méthodes d'approximation utilisant des méthodes de Monte Carlo** pour estimer les probabilités en chaque noeud connaissant la probabilité de certains noeuds.

12.2.2 La d-séparation généralisée

La d-séparation exploitée au niveau local des noeuds d'un graphe dans la section 12.1 (voir équation 12.3) peut être généralisée au calcul de l'indépendance de sous-ensembles de noeuds dans le graphe.

Donnons d'abord une définition étendue de l'indépendance conditionnelle :

Définition 12.2

Deux variables aléatoires X et Y sont conditionnellement indépendantes sachant Z , avec Z un ensemble de variables aléatoires si et seulement si :

$$P(X, Y | \mathcal{Z}) = P(X | \mathcal{Z})P(Y | \mathcal{Z})$$

Le fait de connaître s'il existe de telles relations entre les variables va permettre d'accélérer les calculs d'inférence.

L'indépendance conditionnelle peut aussi se décrire dans un réseau bayésien en terme de géométrie du graphe. Un théorème [Pea88] énonce en effet que :

Théorème 12.2

Deux variables X et Y sont conditionnellement indépendantes sachant un ensemble de variables \mathcal{Z} si et seulement si les noeuds X et Y sont d-séparables par l'ensemble \mathcal{Z} .

Il faut maintenant définir la géométrie de *d-séparabilité* dans un graphe.

Soient deux noeuds X et Y et un ensemble de noeuds \mathcal{Z} , avec $X \notin \mathcal{Z}$ et $Y \notin \mathcal{Z}$. Nous allons d'abord définir ce qu'est un *chemin non orienté* entre X et Y . Pour cela, il faut imaginer un instant que chaque arc du graphe est doublé d'un autre arc, orienté dans l'autre sens. Si on peut aller de X à Y dans ce nouveau graphe, on dit alors que X et Y sont reliés par un chemin non orienté. Si l'on préfère, cela revient à autoriser à emprunter les arcs à l'envers, en plus de leur sens de parcours naturel.

Sur un chemin non orienté entre X et Y , on peut rencontrer quatre types de noeuds : deux sortes de noeuds *en série*, des noeuds *convergents* et des noeuds *divergents*. En effet, pour un noeud intermédiaire Z , le noeud le plus proche en direction de X peut être soit un fils, soit un père ; de même en direction de Y . La figure 12.4 montre ces différents cas.

Ceci posé, nous pouvons maintenant caractériser un chemin non orienté entre X et Y vis à vis d'un ensemble de noeuds \mathcal{Z} du graphe tel que ni X , ni Y n'appartiennent à \mathcal{Z} . Nous distinguons les cas suivants :

Cas 1 Z est dans \mathcal{Z} et Z est un noeud divergent.

Cas 2 et 3 Z est dans \mathcal{Z} et Z est un noeud en série.

Cas 4 Soit $\mathcal{T}(X, Y)$ l'ensemble des noeuds qui sont descendants à la fois de X et de Y , mais dont aucun des parents ne possède la même propriété. Alors, aucun des noeuds $T \in \mathcal{T}(X, Y)$ ni aucun des descendants de T n'est inclus dans \mathcal{Z} .

On a alors la définition :

Définition 12.3

Soit \mathcal{Z} un ensemble de noeuds et X et Y deux noeuds n'appartenant pas à \mathcal{Z} . On dit que \mathcal{Z} d-sépare X et Y si pour tous les chemins non orientés entre X et Y , il existe un noeud Z dans \mathcal{Z} dans un des des cas 1, 2 ou 3, et si le cas 4 est vrai pour tous les noeuds $T \in \mathcal{T}(X, Y)$.

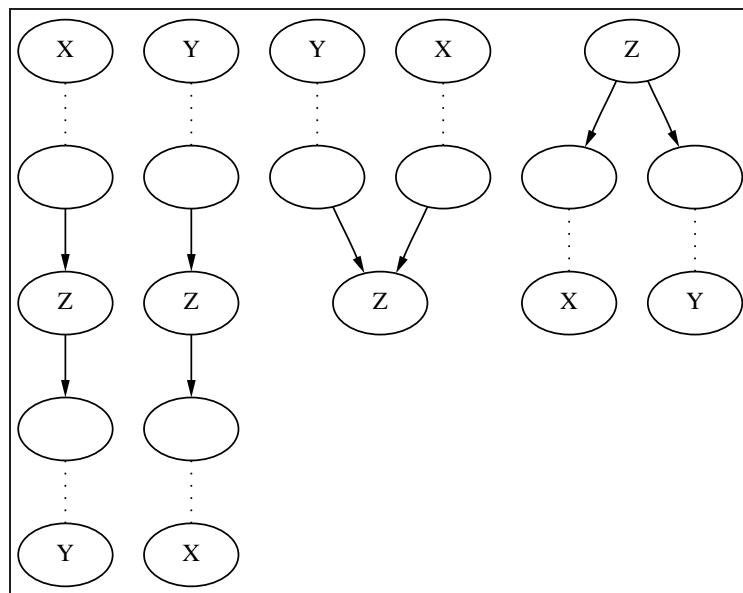


FIG. 12.4 – Les différents types de nœuds sur un chemin non orienté entre X et Y : les deux premiers sont en série, le troisième est convergent et le quatrième divergent. Les pointillés non fléchés représentent un sous-chemin non orienté.

L'équivalence entre l'indépendance conditionnelle et la d -séparabilité peut s'étendre à des ensembles de nœuds par la définition suivante :

Définition 12.4

On dit que deux ensembles de nœuds \mathcal{X} et \mathcal{Y} sont d -séparés par un ensemble de nœuds \mathcal{Z} si

$$\forall X \in \mathcal{X}, \forall Y \in \mathcal{Y} : X \text{ et } Y \text{ sont } d\text{-séparés par } \mathcal{Z}$$

On a alors le théorème étendu :

Théorème 12.3

\mathcal{X} et \mathcal{Y} sont d -séparés par \mathcal{Z} si et seulement si \mathcal{X} et \mathcal{Y} sont conditionnellement indépendants sachant \mathcal{Z} : $P(\mathcal{X}, \mathcal{Y} | \mathcal{Z}) = P(\mathcal{X} | \mathcal{Z})P(\mathcal{Y} | \mathcal{Z})$.

Exemple 9 (Retour sur l'exemple)

En reprenant l'exemple 12.3, on constate par exemple que SA et ZO sont conditionnellement indépendantes sachant FN . Ces deux nœuds sont en effet d -séparables par $\mathcal{Z} = \{FN\}$. Le nœud $FN \in \mathcal{Z}$ est sur le chemin non orienté $SA \rightarrow FN \rightarrow CP \rightarrow ZO$; c'est un nœud divergent.

Utilisation de la d -séparation

Le calcul des sous-ensembles de nœuds conditionnellement indépendants permet ainsi de simplifier les calculs d'inférence. Lorsque l'on cherche à maximiser les indépendances conditionnelles dans un graphe, on arrive à la notion de polyarbres.

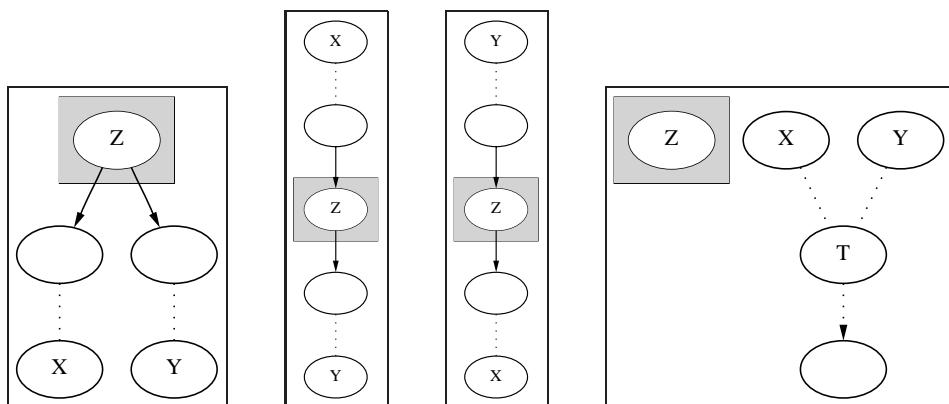


FIG. 12.5 – Les quatre configurations servant à définir la d -séparation. La zone griseée représente l’ensemble \mathcal{Z} . Les pointillés fléchés représentent un sous-chemin orienté.

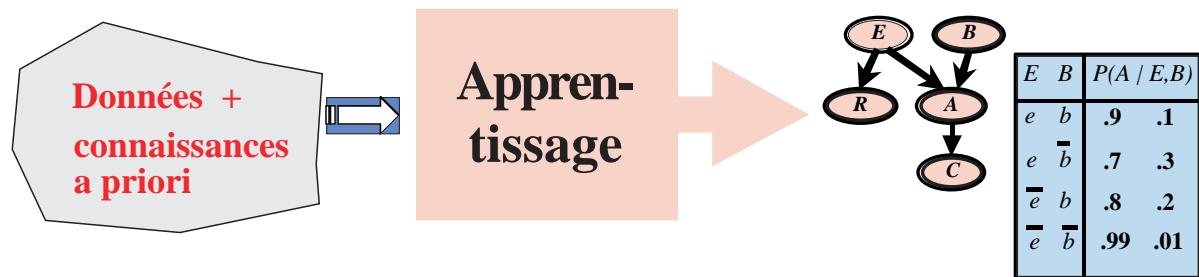


FIG. 12.6 – Un réseau bayésien comprend à la fois une structure et des paramètres associés aux nœuds de cette structure (les probabilités conditionnelles). L’apprentissage consiste à estimer les paramètres et parfois aussi la structure à partir de données et éventuellement de connaissances préalables.

12.3 L’apprentissage des réseaux bayésiens

L’apprentissage consiste à trouver un réseau bayésien modélisant les données disponibles en s’appuyant éventuellement sur les connaissances *a priori* disponibles (voir la figure 12.6). Quatre grandes familles de problèmes sont rencontrées qui correspondent à des classes de méthodes d’apprentissage spécifiques.

	Structure connue	Structure inconnue
Données complètes	Estimation statistique paramétrique	Optimisation discrète sur les structures (algorithmes de recherche discrète)
Données incomplètes	Optimisation paramétrique (EM, descente de gradient,...)	Combinaison de méthodes (EM structurelle, mélange de modèles,...)

Nous examinons tour à tour ces différents types de problèmes.

12.3.1 Apprentissage avec structure connue et données complètes

Dans le cas où la structure du réseau est connue (grâce à un expert par exemple), le problème est d'estimer les paramètres de ce réseau, c'est-à-dire les tables de probabilités conditionnelles en chaque nœud. Il s'agit de trouver le réseau le « plus proche » de la loi de probabilité ayant engendré les données. Cette tâche est proche de celle consistant à estimer un paramètre Θ permettant de modéliser les données $\mathcal{S} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m)$ par une distribution: $p(\mathcal{S}|\Theta)$.

En général, le choix du paramètre dépend du choix de la famille de distributions de probabilités : multinomiale, Gaussienne, de Poisson, etc. Mais le principe est toujours celui de chercher la valeur du paramètre Θ maximisant la fonction de vraisemblance⁴:

$$\mathcal{L}(\Theta : \mathcal{S}) = P(\mathcal{S}|\Theta) = \prod_{i=1}^m P(\mathbf{z}_i|\Theta)$$

Dans le cas d'un réseau bayésien, la vraisemblance s'écrit en prenant en compte les indépendances conditionnelles des nœuds n_j en fonction de leurs parents Pa_j . On aura ainsi pour m données \mathbf{x}_i , chacune étant décrite sur d attributs (soit ici d nœuds du réseau):

$$\begin{aligned} \mathcal{L}(\Theta : \mathcal{S}) &= \prod_{i=1}^m P(\mathbf{x}_i(1), \dots, \mathbf{x}_i(d) : \Theta) = \prod_{i=1}^m \prod_{j=1}^d P(\mathbf{x}_i(j)|Pa_j : \Theta_j) \\ &= \prod_{j=1}^d \prod_{i=1}^m P(\mathbf{x}_i(j)|Pa_j : \Theta_j) = \prod_{j=1}^d \mathcal{L}_j(\Theta_j : \mathcal{S}) \end{aligned}$$

où $\mathbf{x}_i(j)$ représente la j ème coordonnées de la donnée \mathbf{x}_i .

La vraisemblance se décompose selon la structure du réseau. De ce fait, on a affaire maintenant à plusieurs problèmes indépendants d'estimation de vraisemblance: le problème est factorisé. Si les paramètres de chaque famille ne sont pas reliés, ils peuvent être estimés indépendamment les uns des autres.

Nous avons examiné ci-dessus le problème comme celui de l'estimation de la valeur la plus probable du paramètre Θ : il s'agit là de l'approche du *maximum de vraisemblance*. Nous pourrions aussi envisager ce problème comme celui de l'estimation de densité θ et non plus de sa valeur la plus probable. C'est l'approche de l'*estimation bayésienne*. Nous n'en donnons pas le détail dans le cas présent, mais le principe général est décrit dans le chapitre 17.

Dans tous les cas, il faut avoir recours à des fréquences calculées pour estimer les probabilités conditionnelles. Nous illustrons cela ci-dessous.

Exemple 10 (Estimation de probabilités conditionnelles par fréquence)

On suppose ici posséder la structure du réseau et chercher à estimer les probabilités conditionnelles nécessaires. Pour réaliser cette estimation, on dispose d'un certain nombre d'observations sur les variables, c'est-à-dire d'exemples de son comportement.

4. Des détails sur le principe du maximum de vraisemblance figurent dans les chapitres 2 et 14.

Reprenons l'exemple de la Figure 12.3, en supposant disposer de la table suivante des observations :

SA	CP	FN	ZO	Nombre d'exemples
VRAI	VRAI	VRAI	VRAI	54
VRAI	VRAI	VRAI	FAUX	1
VRAI	FAUX	VRAI	VRAI	7
VRAI	FAUX	VRAI	FAUX	27
FAUX	VRAI	VRAI	VRAI	3
FAUX	FAUX	VRAI	FAUX	2
FAUX	VRAI	FAUX	VRAI	4
FAUX	FAUX	FAUX	FAUX	2
				100

Prenons le nœud CP du réseau, dont les parents sont FN et ZO . Nous cherchons à estimer les huit valeurs $P(CP|FN, ZO)$, $P(CP|\neg FN, ZO)$, ..., $P(\neg CP|\neg FN, \neg ZO)$. Par exemple, $P(CP|FN, \neg ZO)$ sera estimée en comptant dans la table le nombre d'exemples pour lesquels CP est *VRAI*, FN est *VRAI* et ZO est *FAUX*, divisé par le nombre d'exemples pour lesquels FN est *VRAI* et ZO est *FAUX*. Cette estimation s'écrit donc : $\hat{P}(CP|FN, \neg ZO) = \frac{1}{1+27+2} = 0.033$

On trouve ainsi, pour compléter :

$\hat{P}(FN)$	$(54 + 1 + 7 + 27 + 3 + 2)/100 = 0.94$
$\hat{P}(ZO)$	$(54 + 7 + 4 + 4)/100 = 0.69$
$\hat{P}(SA FN)$	$(54 + 1 + 7)/(54 + 1 + 7 + 27 + 3 + 2) = 0.66$
$\hat{P}(SA \neg FN)$	$0/(4 + 2) = 0.0$
$\hat{P}(CP FN, ZO)$	$54/(54 + 7 + 3) = 0.84$
$\hat{P}(CP FN, \neg ZO)$	$1/(1 + 27 + 2) = 0.033$
$\hat{P}(CP \neg FN, ZO)$	$0/4 = 0.0$
$\hat{P}(CP \neg FN, \neg ZO)$	$0/2 = 0.0$

12.3.2 Apprentissage avec structure inconnue et données complètes

Il se peut que nous ne disposions pas de modèle des indépendances conditionnelles entre variables *a priori*. Il faut alors apprendre à la fois le réseau bayésien encodant ces indépendances et les paramètres associés. L'apprentissage de la structure du réseau est intéressante à plusieurs titres :

- Cela peut conduire à une *meilleure généralisation* à partir des données. En effet, le réseau encode des indépendances, ce qui signifie qu'il y a moins de paramètres à apprendre, et donc un espace d'hypothèses plus contraint.
- Cela permet d'obtenir des propriétés structurales inaccessibles avec d'autres représentations non structurées. On peut ainsi mettre à jour des indépendances, mais aussi des relations de cause à effet entre variables.

Il existe deux grandes familles d'approches pour apprendre la structure d'un réseau bayésien à partir de données :

1. **Les approches basées sur les contraintes.** Le principe en est de tester les indépendances conditionnelles, et de chercher une structure de réseau cohérente avec les dépendances et indépendances observées.
2. **Les approches utilisant une fonction de score.** Dans ces approches, un score est associé à chaque réseau candidat, mesurant l'adéquation des (in)dépendances encodées dans le réseau avec les données. On cherche alors un réseau maximisant ce score.

Ces deux familles d'approches sont bien fondées (du point de vue des statistiques), c'est-à-dire qu'avec suffisamment de données, l'apprentissage converge vers une structure correcte dans les deux cas. Cependant les premières sont sensibles aux erreurs dans les tests d'indépendance, tandis que pour les secondes la recherche d'une structure optimale est un problème NP-difficile.

Les approches utilisant une fonction de score étant les plus utilisées, nous nous concentrerons sur celles-ci dans la suite.

12.3.2.1 Les fonctions de score

Le score naturel pour évaluer une structure est sa vraisemblance. Sans entrer dans les détails de sa dérivation, celle-ci peut s'écrire :

$$\begin{aligned} I(G : \mathcal{S}) &= \log \mathcal{L}((G : \mathcal{S}) \\ &= m \sum_{j=1}^d (I(\mathbf{x}(j) : P_{\mathcal{A}}^G(j)) - H(\mathbf{z}(j))) \end{aligned}$$

où $H(X)$, l'*entropie*, mesure combien X encode d'information et où $I(X; Y)$ est l'*information mutuelle* entre les variables X et Y et mesure l'information que chaque variable fournit sur l'autre variable ($I(X; Y) \geq 0$, $I(X; Y) = 0$ ssi X et Y sont indépendantes, et $I(X; Y) = H(X)$ ssi X est totalement prédictible connaissant Y).

Cette formule est séduisante car elle correspond bien à une mesure intuitive de la qualité du réseau : elle favorise les structures dans lesquelles les variables sont maximalement dépendantes de leurs parents, ce qui est ce que l'on veut.

Malheureusement, ce score conduit à préférer les réseaux trop proches des données car il est toujours meilleur d'ajouter un arc (puisque $I(X; Y) \leq I(X; Y, Z)$). De ce fait, il y a risque de surapprentissage, c'est-à-dire que le réseau encode des corrélations accidentnelles dans les données, qui ne correspondent pas à des régularités vraies. C'est pourquoi il faut utiliser des techniques permettant de combattre ce phénomène, c'est-à-dire permettant de contrôler l'induction (voir les chapitres 2, 3 et 17).

- *Restriction de l'espace d'hypothèses.* Par exemple en limitant le nombre de parents possibles pour chaque nœud ou le nombre de paramètres dans les tables de probabilités conditionnelles.
- *Régularisation par utilisation du principe de description minimale (MDLp).* Ce principe est décrit plus précisément dans le chapitre 17, mais il consiste essentiellement à chercher un compromis entre l'adéquation aux données et la complexité du modèle choisi pour en rendre compte.
- *Estimation bayésienne.* (voir chapitre 17). Il s'agit de faire une moyenne sur toutes les valeurs de paramètres possibles.
- On peut aussi vérifier la qualité du réseau appris en utilisant une *technique de validation* par un ensemble test (voir chapitre 3).
- D'autres techniques de contrôle de l'espace d'hypothèses et de sélection de modèles existent (voir le chapitre 17). Elles sont moins utilisées pour l'apprentissage de réseaux bayésiens.

Le score le plus fréquemment utilisé est celui du *principe de description minimale (MDLp)*. Selon le *MDLp*, il faut choisir le réseau \mathcal{B} tel que la somme de la longueur de description du réseau et celle des données encodées à l'aide de \mathcal{B} soit minimale. Ce principe conduit à chercher un réseau juste assez complexe pour pouvoir raisonnablement décrire les données en l'utilisant. La description d'un réseau \mathcal{B} implique celle du graphe G et celle de l'ensemble des distributions

de probabilités conditionnelles associées P . D'où la formule à minimiser sur \mathcal{B} :

$$L(\mathcal{S}) = L(\mathcal{B}) + L(\mathcal{S}|\mathcal{B}) = L(G) + L(P) + L(\mathcal{S}|\mathcal{B})$$

Pour **décrire le graphe acyclique orienté** G , il suffit de coder pour chaque variable X_j une description de ses parents $Pa(j)$. Pour cela, il faut coder le nombre k de parents et l'index de l'ensemble $Pa(j)$ dans l'énumération de tous les ensembles de taille k parmi n noeuds $\binom{n}{k}$. Comme k peut être codé en utilisant $\log n$ bits et l'index par $\log \binom{n}{k}$ bits, la longueur de description de la structure est :

$$L(G) = \sum_j \left(\log n + \log \binom{n}{|Pa(j)|} \right)$$

où les log sont en base 2 pour obtenir une longueur en bits, et où $\binom{n}{k}$ représente en notation américaine le nombre de combinaisons de n éléments pris k à k .

Pour **décrire** P , il faut coder les paramètres de chaque table de probabilités conditionnelles associées à chaque noeud. Pour la table associée à une variable X_j , il faut coder $\|Pa(j)\|(\|X_j\|-1)$ paramètres. La longueur de description de ces paramètres dépend du nombre de bits utilisés pour coder chaque paramètre numérique. Le choix usuel est $1/2 \log m$ (voir [M.99], p.428). D'où :

$$L(X_j, Pa(j)) = \frac{1}{2} \|Pa(j)\| (\|X_j\| - 1) \log m$$

Pour **décrire les donnés**, on utilise la mesure de probabilité définie par le réseau \mathcal{B} pour construire un code de Huffman pour les exemples dans \mathcal{S} . Dans ce code, la longueur de chaque mot de code dépend de la probabilité assignée à cet exemple. Cette longueur de description est ainsi approximée par :

$$L(\mathcal{S}|\mathcal{B}) = - \sum_{i=1}^m \log P_{\mathcal{B}}(\mathbf{x}_i)$$

que l'on peut réécrire après calculs ([M.99], p.429)) : $m \sum_j H(X_j|Pa(j))$

D'où l'expression finale de la longueur totale de description associée à un réseau \mathcal{B} et des données \mathcal{S} :

$$L(\mathcal{S}) = \sum_j \left(\log n + \log \binom{n}{|Pa(j)|} \right) + \frac{1}{2} \|Pa(j)\| (\|X_j\| - 1) \log m + m \sum_j H(X_j|Pa(j))$$

qui a l'avantage d'être décomposable et donc de se prêter à des méthodes de recherche locale, où l'on cherche à améliorer le score par des modifications de chaque noeud.

12.3.2.2 L'apprentissage avec une fonction de score

Une fois un score défini sur les réseaux, l'apprentissage consiste à trouver le réseau minimisant ce score. Malheureusement, les résultats actuels suggèrent qu'il s'agit là d'un problème NP-difficile. En effet, le nombre de graphes est plus qu'exponentiel en fonction du nombre (donné) de variables. Il faut donc avoir recours à des techniques d'exploration heuristiques. Nous ne discutons ici que de la plus simple: la *descende de gradient*.

La démarche consiste à démarrer avec un réseau (souvent le réseau vide) et à appliquer itérativement sur le réseau candidat courant l'opérateur (par exemple *ajout* ou *retrait* ou *inversion* d'arc) conduisant au meilleur accroissement du score. Cette procédure est répétée jusqu'à

ce qu'un maximum (local) soit atteint. Les résultats expérimentaux montrent que cette technique est souvent très efficace malgré son caractère myope et glouton. L'un des problèmes est de vérifier que l'on conserve à chaque pas un graphe acyclique orienté. Le choix de la direction d'un arc est également un problème non trivial à résoudre.

D'autres techniques incluent la recherche tabou, le recuit simulé, etc. (voir le chapitre 3).

12.3.3 Apprentissage en présence de données incomplètes

Il arrive fréquemment que les données réelles soient incomplètes, c'est-à-dire que des variables utiles ne soient pas mesurées. Cela peut se produire en raison de valeurs manquantes (par exemple tous les patients rentrant à l'hôpital ne subissent pas les mêmes examens), mais aussi parce que des variables sont inobservables ou insoupçonnées comme c'est le cas lorsqu'un phénomène est mal connue et que des variables causales sous-jacentes sont ignorées. On parle alors de *variables latentes* ou de variables cachées.

Le problème est que l'absence de ces variables non seulement obscurcit la compréhension des dépendances propres au domaine, mais peut également conduire à apprendre trop de paramètres comme le montre la figure 12.7.

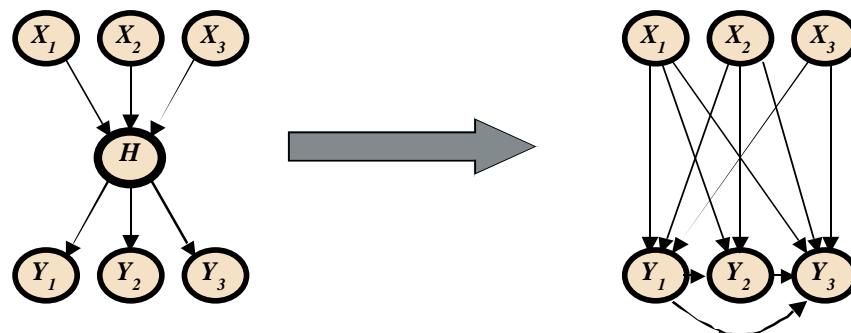


FIG. 12.7 – L'absence d'une variable peut conduire à avoir à estimer 59 paramètres (à droite) au lieu de 17 (à gauche). (D'après un tutoriel de Nir Friedman).

La recherche de variables latentes est l'un des plus gros problèmes en apprentissage de réseaux bayésiens.

12.3.4 Apprentissage avec structure connue et données incomplètes

La différence essentielle avec l'apprentissage de paramètres en présence de données complètes est que le problème de trouver les paramètres correspondant au maximum de vraisemblance est qu'il s'agit maintenant d'un problème d'optimisation non linéaire avec potentiellement de très nombreux optima locaux. Les méthodes en descente de gradient deviennent donc plus problématiques. On utilise généralement une méthode *EM* (Expectation-Maximization). (Voir figure 12.8).

La méthode *EM* est décrite dans l'annexe 18.9. Nous l'illustrons ici sur l'exemple vu plus haut.

Exemple 11 (Méthode *EM* pour les réseaux bayésiens)

Reprendons l'exemple précédent en supposant que les informations que nous possédons sont incomplètes, c'est à dire que certaines instanciations des variables sont inconnues. Dans cet

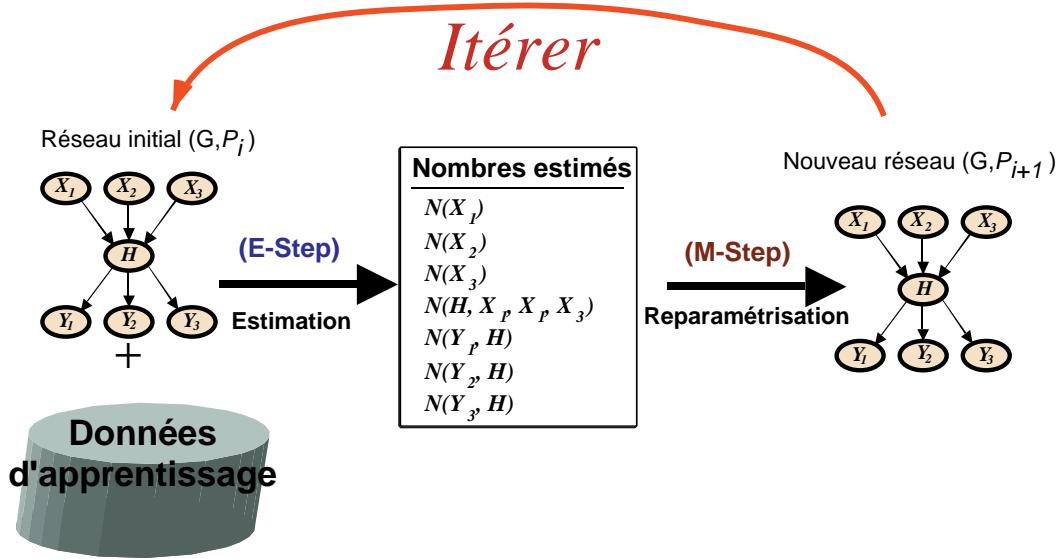


FIG. 12.8 – *Principe général de la méthode EM. (D'après un tutoriel de Nir Friedman).*

exemple, trois valeurs manquent : elles sont indiquées par « ? ».

SA	CP	FN	ZO	Nombre d'exemples
VRAI	VRAI	VRAI	VRAI	54
VRAI	VRAI	VRAI	FAUX	1
?	?	VRAI	VRAI	7
VRAI	FAUX	VRAI	FAUX	27
FAUX	VRAI	?	VRAI	3
FAUX	FAUX	VRAI	FAUX	2
FAUX	VRAI	FAUX	VRAI	4
FAUX	FAUX	FAUX	FAUX	2
				100

Est-il encore possible de réaliser une estimation rationnelle des probabilités conditionnelles caractéristiques ? La réponse (positive) est donnée par l'utilisation de l'algorithme *EM* (voir l'annexe 18.9). Mais il faut d'abord transformer un peu le problème.

Prenons les trois exemples pour lesquels il manque la valeur de *B*. Dans le tableau des données, en tenant compte de ce que $P(\neg FN | \neg SA, CP, ZO) = 1 - P(FN | \neg SA, CP, ZO)$, on peut les remplacer par les « exemples virtuels » suivants :

SA	CP	FN	ZO	Nombre d'exemples
FAUX	VRAI	VRAI	VRAI	$3 \times P(\neg FN \neg SA, CP, ZO)$
FAUX	VRAI	VRAI	VRAI	$3 \times (1 - P(FN \neg SA, CP, ZO))$
				100

Ceci appelle deux remarques : d'abord que le « nombre » de certains exemples n'est plus entier ; mais cela ne gêne pas les calculs. Ensuite, que la valeur $P(\neg FN | \neg SA, CP, ZO)$ est inconnue et que pour la calculer par les techniques d'inférence, il faudrait connaître toutes les probabilités conditionnelles caractéristiques du réseau.

Cette seconde objection paraît insurmontable ; en réalité, l'algorithme *EM* est capable d'estimer itérativement la valeur cachée $P(\neg FN | \neg SA, CP, ZO)$. La technique est la suivante. Supposons cette valeur connue ; on peut compléter l'ensemble d'apprentissage comme ci-dessus, par des exemples virtuels. Cet ensemble étant maintenant complet, on peut appliquer les méthodes

d'inférence du paragraphe précédent pour estimer les probabilités conditionnelles caractéristiques du réseau... parmi laquelle se trouve la valeur inconnue $P(\neg FN|\neg SA, CP, ZO)$. Cette nouvelle valeur sert à fabriquer un nouvel ensemble d'apprentissage, et ainsi de suite jusqu'à stabilisation de la valeur inconnue.

Le même raisonnement peut s'appliquer aux deux valeurs manquantes des sept exemples pour lesquels on a $FN = VRAI$ et $ZO = VRAI$. Mais ici, il y a deux valeurs cachées⁵: $P(SA, CP|FN, ZO)$ et $P(SA, \neg CP | FN, ZO)$ ⁶.

La raison de la convergence de l'algorithme *EM* est esquissée dans l'annexe 18.9, mais la preuve complète sur ce cas particulier des réseaux bayésiens est hors du champ de cet ouvrage. D'autres exemples sont également développés dans cette annexe. D'autres applications de cet algorithme à l'apprentissage se trouvent au chapitre 13, qui traite des *Modèles de Markov cachés* et au chapitre 15, dont le sujet est la classification non supervisée.

12.3.5 Apprentissage avec structure inconnue et données incomplètes

L'apprentissage de structure en présence de données incomplètes conjugue tous les problèmes examinés plus haut. Pour le moment, il s'agit surtout d'un domaine de recherche avec des méthodes encore en cours de conception et d'examen.

L'approche la plus simple est de combiner les méthodes utilisées pour l'apprentissage de paramètres et l'apprentissage de structure. L'idée est pour chaque structure G d'estimer les paramètres optimaux P en utilisant soit une technique de descente de gradient, soit une technique *EM*, puis une fois cela fait, on associe un score à G . On examine alors les graphes obtenus à partir de G avec des opérateurs de changement de structure, et on choisit celui ayant le meilleur score.

Le problème majeur de cette approche est son coût computationnel très élevé. Elle n'est praticable que pour de très petits réseaux.

Une autre approche a été proposée utilisant une généralisation de la méthode *EM*: l'*EM-structurel*. L'idée est d'utiliser les paramètres trouvés pour les structures précédentes pour aider à évaluer de nouvelles structures portant sur le même ensemble de variables aléatoires.

Le principe général en est le suivant (voir la figure 12.9) :

1. Effectuer une recherche dans l'espace des structures et des paramètres.
2. Utiliser des itérations à la *EM*, en employant la meilleure solution précédemment trouvée comme d'une base pour soit :
 - trouver les meilleurs paramètres (en fonction du score) : *EM* « paramétrique »
 - trouver la meilleure structure (en fonction du score) : *EM* « structurel ».

Réalisations industrielles

Les réseaux bayésiens possèdent plusieurs propriétés qui les rendent intéressants pour des applications :

- Ils s'adaptent sans problème aux bases de données incomplètes.
- Ils sont conçus pour rendre compte de relations causales.
- Ils permettent d'intégrer des connaissances du domaine et des données plus facilement que beaucoup d'autres techniques.

5. Pour un exemple ayant K valeurs inconnues, il y a 2^{K-1} valeurs cachées.

6. Il y a en réalité quatre probabilités conditionnelles inconnues, mais les deux autres, $P(\neg SA, CP|FN, ZO)$ et $P(\neg SA, \neg CP|FN, ZO)$ se déduisent des deux premières.

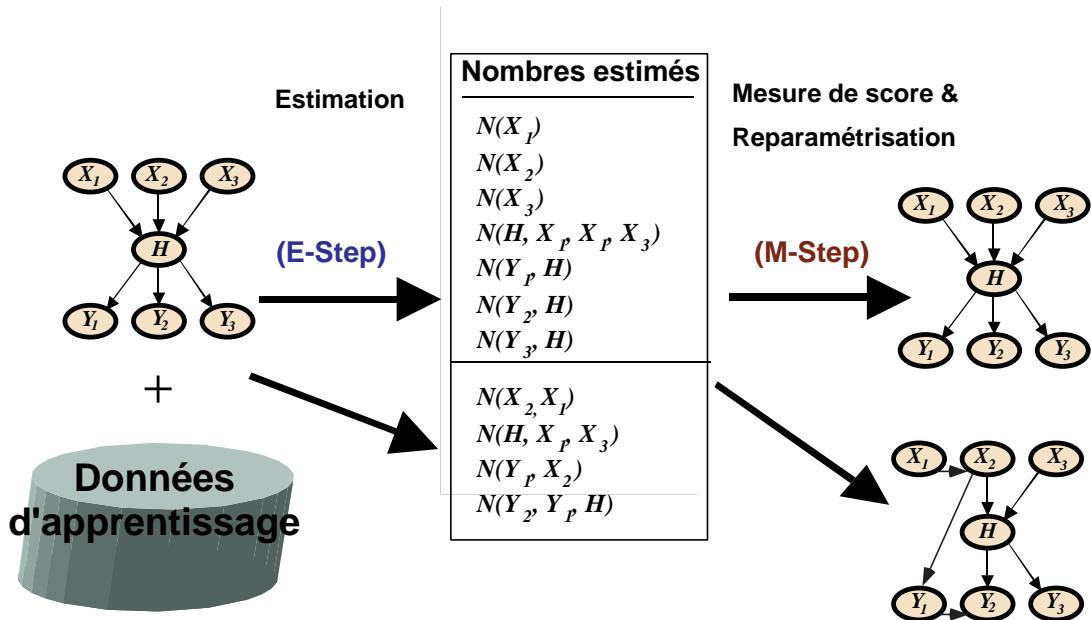


FIG. 12.9 – Principe général de la méthode EM-structurel. (D'après un tutoriel de Nir Friedman).

- Par leur parenté avec les approches d'induction bayésienne, ils sont mieux armés pour éviter les problèmes de surapprentissage (voir le chapitre 3).

Si chacun de ces points peut être soumis à discussion, il n'en reste pas moins que les réseaux bayésiens sont des modèles très séduisants.

Nous reprenons ici brièvement, en particulier, quelques unes des applications que détaillent Becker et Naim ([BN99], chapitre 9). Peu utilisent les fonctions d'apprentissage : il est certain que, compte tenu du nombre déjà grand d'applications, cet aspect ne peut qu'être appelé à se développer. Pour donner un ordre de grandeur, certains réseaux fonctionnent actuellement avec plusieurs milliers de noeuds⁷.

- La compagnie AT&T a mis en place un réseau bayésien pour la détection des mauvais payeurs, à partir d'une base de données de plusieurs millions d'appels.
- La NASA utilise un système graphique pour suivre en temps réel les paramètres de contrôle des moteurs.
- La société Ricoh utilise des réseaux bayésiens pour le diagnostic de ses appareils de photocopie.
- L'investissement de Microsoft dans les réseaux bayésien est important, tant en ce qui concerne le développement de logiciels que la recherche. Les applications visées sont : le diagnostic sur les réseaux, l'aide à l'utilisateur (le « trombone » d'Office en est en exemple), la validation de gros logiciels, etc.
- Et beaucoup d'applications dans le domaine de la santé, du militaire, etc.

7. Un réseau a été mis au point à Stanford pour modéliser le fonctionnement de la cellule. Il comporte vingt-deux millions de noeuds !

Notes historiques et sources bibliographiques

Les réseaux bayésiens sont nés des travaux de J. Pearl ([Pea88]). Il est intéressant de noter que dans son dernier ouvrage ([Pea00]), il considère cette technique comme mal fondée théoriquement. Les années 1990 ont vu le développement des théories et des réalisations, ces dernières en très grand nombre (voir ci-dessous). Les concepts les plus amont regroupent les HMM, les réseaux bayésiens et les réseaux connexionnistes sous le terme général de « modèles graphiques ». Il est possible que cette vision très générale permette d'envisager des algorithmes d'apprentissage plus puissants, mais cette unification théorique ne semble pas avoir encore porté de fruits, sur le plan algorithmique du moins. Les réseaux bayésiens sont actuellement l'objet de gros projets de recherche et de développement. Ces outils sont encore loin d'avoir montré toutes leurs possibilités.

Le texte de ce chapitre a été fortement inspiré par l'ouvrage remarquable (et pas seulement sur ce sujet) de N. Nilsson ([Nil98]). Le livre d'Ann Becker et Patrick Naïm [BN99] est une introduction très recommandable aux principes et aux applications de ces outils.

Résumé

- Les réseaux bayésiens sont des modèles permettant de décrire les relations de probabilités conditionnelles entre des faits. Cette représentation repose sur un graphe orienté sans cycle (DAG) dans lequel chaque nœud, c'est-à-dire chaque variable du monde modélisé, possède une table de probabilités conditionnelles, et où chaque arc représente une dépendance directe entre les variables reliées. Ces réseaux représentent alors la distribution de probabilités jointes de l'ensemble des variables de manière compacte, en s'appuyant sur les relations d'indépendance conditionnelle.
- Moyennant une propriété locale d'indépendance, il est possible d'effectuer le calcul de la probabilité de tout groupe de faits connaissant tout autre groupe.
- L'apprentissage automatique des valeurs des probabilités conditionnelles peut se faire à partir d'un ensemble d'apprentissage, même incomplet, si la structure du réseau est donnée.
- Il existe aussi des techniques pour apprendre à la fois l'architecture et les probabilités conditionnelles définissant complètement un réseau bayésien. On recense deux grandes familles de méthodes d'apprentissage de réseaux bayésiens : celles utilisant les indépendances conditionnelles pour construire le réseau et celles définissant un score permettant de guider la recherche d'un réseau en accord avec les données.

Chapitre 13

L'apprentissage de modèles de Markov cachés

Quand les objets sur lesquels porte l'apprentissage sont des séquences d'événements, le concept extrait doit refléter à la fois la nature de ces événements et la manière dont ils s'enchaînent. On a déjà vu au chapitre 2 et au chapitre 3 des exemples d'objets de structure séquentielle. On a vu aussi au chapitre 7 des méthodes pour extraire des concepts sous forme de grammaires à partir d'exemples et de contre-exemples. Nous présentons dans ce chapitre un outil puissant pour induire un concept de nature statistique à partir seulement de séquences d'apprentissage appartenant à ce concept : les modèles de Markov cachés, ou HMM¹.

*Par leur nature statistique, les HMM se situent facilement dans le cadre de la décision bayésienne, qui a été présenté au chapitre 2. En particulier, le principe du maximum de vraisemblance *a posteriori* (MAP) prescrit d'attribuer une séquence inconnue à la classe qui a la plus grande probabilité de l'avoir engendrée. L'apprentissage consiste donc dans ce cadre à apprendre pour chaque classe de séquences le HMM le plus vraisemblable. En pratique, le problème revient à apprendre indépendamment un HMM par classe, sans tenir compte des contre-exemples.*

1. En anglais : *Hidden Markov Models*.

REVENONS sur l'étang où nagent des oies et des cygnes. L'ornithologue que nous avons connu débutant au commencement de ce livre est maintenant plus expérimenté. Ce matin, il arrive très tôt pour observer les oiseaux se poser. La veille, il était venu dans la matinée, quand tous les animaux étaient là et il avait observé une trentaine de cygnes et environ quatre-vingt oies. Il espère donc voir arriver une bonne centaine d'oiseaux.

De fait, les vols commencent et il y a bientôt sur le lac les premiers oiseaux. Mais pas dans la proportion attendue : vingt cygnes et quatre oies se sont d'abord posés. Dans les minutes qui suivent, une dizaine d'oiseaux se posent, moitié oies, moitié cygnes. Finalement, arrive le reste de la troupe des oies, soit environ soixante-dix éléments, et de temps en temps un cygne. Au total, ces derniers sont finalement une trentaine.

L'observateur comprend alors que les deux espèces n'ont pas les mêmes habitudes : les cygnes sont plus matinaux, ce qui explique que la proportion entre les deux espèces varie énormément dans le temps d'arrivée. En notant **O** l'arrivée d'une oie et **C** celle d'un cygne, et en mettant un intervalle entre les trois phases d'arrivée, la séquence observée peut se dénoter ainsi :

Comment décrire ce phénomène ? Attention : il ne s'agit pas seulement d'apprendre cette séquence par cœur. Il faut tenir compte du fait que l'ordre exact d'arrivée des oiseaux ne se reproduira pas exactement à l'identique chaque matin : certains oiseaux ne viendront pas, certains voleront plus ou moins vite, etc. Si l'ornithologue veut expliquer ce qu'il observe et prédire ce qu'un autre pourra observer, il doit donc faire l'apprentissage d'un concept qui décrive de manière satisfaisante les propriétés de telles séquences.

Si cet avimateur a lu le chapitre qui suit, il produira peut-être un concept exprimé sous la forme du tableau et du graphique (figure 13.1) qui sont donnés ci-dessous.

	Probabilité d'observer un cygne	Probabilité d'observer une oie
Période 1	0.8	0.2
Période 2	0.5	0.5
Période 3	0.1	0.9

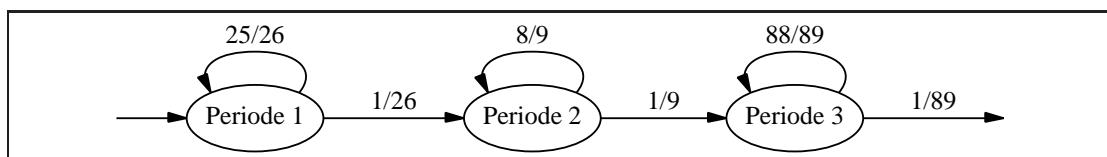


FIG. 13.1 – *Comment les cygnes et les oies arrivent sur l'étang.*

Comment interpréter ce modèle? Le cercle étiqueté « période 1 » correspond à la phase d'arrivée majoritaire des cygnes, celui étiqueté « période 2 » au moment où les populations sont en fréquence d'arrivée égale, et le dernier à l'arrivée massive des oies (avec quelques cygnes parmi elles). La succession temporelle se traduit par un parcours de gauche à droite en suivant les flèches, avec la règle que chaque passage dans un état correspond exactement à l'observation d'un oiseau, cygne ou oie. Quand on est dans un cercle (appelons-le désormais un état), on a deux solutions: soit y rester en faisant une boucle locale, soit passer au suivant. Le passage d'un état à lui-même ou au suivant est commandé par le chiffre situé sur la flèche, qui est une

probabilité. Par exemple, dans l'état 1, la probabilité de passer à l'état 2 est de 1/26, celle de rester dans l'état 1 est de 25/26.

Et les oiseaux ? Leur observation est commandée par la table donnée au-dessus du graphe des états. Chaque passage dans l'état 1 correspond avec une probabilité de 0.8 à l'observation d'un cygne et donc de 0.2 à celle d'une oie. Dans l'état 2, l'observation est équiprobable. Quand on est dans l'état 3, il est 9 fois plus probable d'observer une oie qu'un cygne.

Faisons maintenant un petit calcul. Combien d'oiseaux sont en moyenne observés pendant le séjour dans l'état 1 ? Environ 25, selon une formule simple du calcul des probabilités². Par conséquent, compte tenu des proportions affectées par le tableau, on observera en moyenne $0.8 \times 25 = 20$ cygnes et $0.2 \times 25 = 5$ oies durant la première période représentée par cet état. Un calcul analogue montre que la durée moyenne de séjour dans l'état 2 est d'environ 8 : on y observera donc (en moyenne) 4 cygnes et 4 oies. Finalement, comme la probabilité de bouclage dans l'état 3 est la plus forte, on y reste en moyenne plus longtemps (le calcul donne 88) et on observe, toujours en moyenne, 80 oies et 8 cygnes.

Au total, la séquence moyenne engendrée par ce modèle comporte une trentaine de cygnes et presque trois fois plus d'oies, avec une proportion d'arrivées des cygnes beaucoup plus forte au début qu'à la fin.

Comme nous allons le voir, le concept décrit ci-dessus est un cas particulier de modèle de Markov caché (HMM). Dans un tel modèle, une séquence est donc considérée comme une suite temporelle gérée par ses états. À chaque instant, un nouvel événement de la séquence est analysé. La théorie des HMM décrit comment passer d'état en état à l'aide de probabilités de transitions et comment chaque élément de la séquence peut être émis par un état du HMM, à l'aide de probabilités d'observation par état. Il permet aussi de calculer la probabilité qu'une séquence donnée ait été émise par un HMM donné.

Les méthodes HMM sont robustes et fiables grâce à l'existence de bons algorithmes d'apprentissage ; de plus, la règle de décision est rapide à appliquer.

2. Si x est la probabilité de boucler dans un état et $1 - x$ celle d'en sortir, la durée moyenne de séjour dans cet état vaut $\frac{x}{(1-x)}$.

Notations utiles pour le chapitre

n	Le nombre d'états du modèle de Markov caché, ou HMM
$S = \{s_1, s_2, \dots, s_n\}$	Les états du HMM
M	La taille de l'alphabet des observations quand celles-ci sont de nature discrète
$V = \{v_1, v_2, \dots, v_M\}$	L'alphabet des observations
A	La matrice des probabilités de transitions entre les états
$a_{ij}, i, j \in [1, n]$	Un élément de A
B	La matrice des probabilités d'observation des symboles de V
$b_j(k), j \in [1, n], k \in [1, M]$	Un élément de B
π	Le vecteur des probabilités initiales du HMM
$\Lambda = (A, B, \pi)$	Un HMM
T	La longueur d'une séquence observée
$O = O_1 \dots O_t \dots O_T$ avec $O_t \in V$	Une séquence observée
$O[i:j] = O_i \dots O_j$	Une sous-séquence de O
$q_1 \dots q_t \dots q_T$ avec $q_t \in S$	La suite des états qui a émis une séquence
$P(O \Lambda)$	La probabilité que le HMM Λ ait émis la séquence O
$\mathcal{O} = O^1 \dots O^m$	Un ensemble d'apprentissage composé de m séquences
$P(\Lambda \mathcal{O})$	La probabilité que l'ensemble de séquences \mathcal{O} ait été émis par le HMM Λ .

13.1 Les modèles de Markov observables

Avant de décrire les HMM proprement dits, nous présentons un modèle probabiliste plus simple pour l'observation de séquences : les *modèles de Markov observables*.

D'une manière générale, un *processus* ou *modèle stochastique observable* est un processus aléatoire qui peut changer d'état s_i , $i = 1, \dots, n$ au hasard, aux instants $t = 1, 2, \dots, T$. Le résultat observé est la suite des états dans lesquels il est passé. On peut aussi dire de ce processus qu'il *émet* des séquences d'états $S = s_1, s_2, \dots, s_T$. Chaque séquence est émise avec une probabilité $P(S) = P(s_1, s_2, \dots, s_T)$. Pour calculer $P(S)$, il faut se donner la probabilité initiale $P(s_1)$ et les probabilités d'être dans un état s_t , connaissant l'évolution antérieure.

Un processus stochastique est *markovien*³ (ou *de Markov*) si son évolution est entièrement déterminée par une probabilité initiale et des probabilités de transitions entre états. Autrement dit, en notant ($q_t = s_i$) le fait que l'état observé à l'instant t est s_i

$$\forall t, \quad P(q_t = s_i | q_{t-1} = s_j, q_{t-2} = s_k, \dots) = P(q_t = s_i | q_{t-1} = s_j)$$

d'où :

$$P(q_1 \dots q_T) = P(q_1) \times P(q_2 | q_1) \times \dots \times P(q_T | q_{T-1})$$

Nous supposons pour simplifier que les processus de Markov auxquels nous avons affaire sont *stationnaires* c'est-à-dire que leurs probabilités de transition ne varient pas dans le temps. Cela autorise à définir une *matrice de probabilité de transitions* $A = [a_{ij}]$ telle que :

$$a_{ij} = P(q_t = s_j | q_{t-1} = s_i) \quad 1 \leq i \leq n, \quad 1 \leq j \leq n$$

3. Au sens strict : markovien *d'ordre 1*.

avec :

$$\forall i, j \quad a_{ij} \geq 0, \quad \forall i \quad \sum_{j=1}^{J=n} a_{ij} = 1$$

Nous appellerons maintenant pour simplifier *modèle de Markov observable* un processus stochastique observable, markovien et stationnaire.

Dans un tel modèle, il y a un lien direct à tout instant entre l'état où se trouve le processus et l'observation faite à cet instant, comme l'illustre la figure 13.2. C'est ce qui caractérise pour nous⁴ le fait que ce processus soit observable. Nous allons maintenant voir comment nous débarasser de cette contrainte en présentant d'autres processus stochastiques : les modèles de Markov cachés. Ensuite, nous comparons leur puissance de modélisation sur un exemple.

13.2 Les modèles de Markov cachés (HMM)

13.2.1 Définition

Le modèle de Markov caché généralise le modèle de Markov observable car il produit une séquence en utilisant deux suites de variables aléatoires ; l'une cachée et l'autre observable.

- La suite cachée correspond à la suite des états q_1, q_2, \dots, q_T , notée $Q(1 : T)$, où les q_i prennent leur valeur parmi l'ensemble des n états du modèle $\{s_1, s_2, \dots, s_n\}$.
- La suite observable correspond à la *séquence des d'observations* O_1, O_2, \dots, O_T , notée $O(1 : T)$, où les O_i sont des lettres d'un alphabet de M symboles observables $V = \{v_1, v_2, \dots, v_M\}$.

Par conséquent, pour un HMM, un état n'est pas associé exclusivement à une lettre donnée qu'il émettrait à coup sûr : chaque lettre a désormais une certaine probabilité d'être émise par chaque état. En outre, ce ne sont pas les états qui sont observés, mais les lettres qu'ils émettent. Une conséquence importante est que l'on peut maintenant travailler avec des alphabets infinis. Une « lettre » est alors émise avec une certaine densité de probabilité, correspondant à une distribution propre à chaque état.

En pratique, on cherche à construire des HMM représentant des concepts dans l'espace de représentation des séquences. Nous prendrons ici pour simplifier des séquences construites sur un alphabet $V = \{v_1, v_2, \dots, v_M\}$ de taille finie. Mais la remarque ci-dessus doit être gardée à l'esprit : la taille de cet alphabet peut être infinie, ce qui signifie en pratique que chaque état peut émettre une variable continue ou un vecteur de \mathbb{R}^d .

13.2.2 Pourquoi faut-il des variables cachées ?

Montrons sur l'exemple de l'introduction la différence entre le modèle de Markov observable et le modèle de Markov caché. Quand on observe l'arrivée des oiseaux sur un étang, on obtient une suite sur l'alphabet $V = \{\text{O}, \text{C}\}$. Une séquence observée sera par exemple :

$$O = \text{O} \text{ O} \text{ C} \text{ O} \text{ C} \text{ O}$$

Les probabilités *a priori* d'observer un cygne ou une oie peuvent être différentes. La construction de deux types de modèles de Markov pour modéliser les séquences sur V va nous conduire à préciser un certain nombre d'éléments relatifs à la nature des états, à leur nombre ainsi qu'aux probabilités de transition et d'observation. Un modèle de Markov observable pour ce problème est représenté dans la Figure 13.2.

4. Si la même observation peut être affectée à plusieurs états, on peut améliorer la capacité de représentation des modèles observables. Nous ne discutons pas cette possibilité ici.

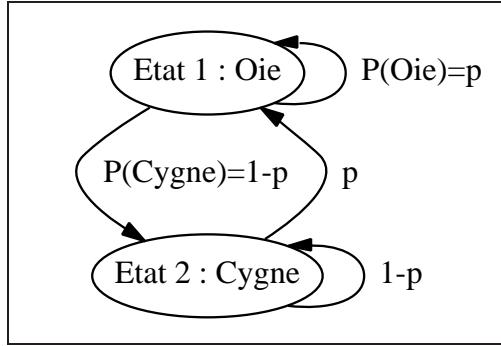


FIG. 13.2 – Le modèle de Markov observable qui modélise la suite des observations des oies et des cygnes.

Il est composé de deux états ; chacun correspond directement à une observation possible: Oie (0) ou Cygne (C). Dans ce modèle, la suite d'états associée à une séquence observée est facile à déterminer : l'observation de 0 correspond à l'état 1 et l'observation de C correspond à l'état 2. Si la probabilité d'observer 0 à l'état 1 est $p = P(Oie)$, alors la probabilité d'observer C à l'état 2 est $1 - p$. La probabilité d'observer la séquence $O(1 : 6) = 0 \ 0 \ C \ 0 \ C \ 0$ se calcule facilement ; elle vaut :

$$p \ p \ (1 - p) \ p \ (1 - p) \ p = p^4 \ (1 - p)^2$$

Elle est par conséquent indépendante de l'ordre d'arrivée des oiseaux et ne tient compte que de leur nombre dans la séquence. Ce modèle n'exprime que les probabilités d'apparition *a priori* des observations.

La figure 13.3, accompagnée du tableau 13.1 définit un modèle de Markov caché (HMM) à deux états.

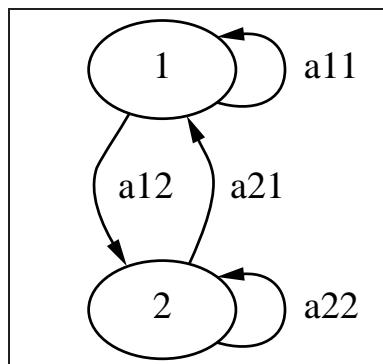


FIG. 13.3 – Le HMM à deux états.

Etat	1	2
P(0)	p1	p2
P(C)	1-p1	1-p2

TAB. 13.1 – Les probabilités d'émission du HMM à deux états.

Sans entrer encore dans les détails, on voit qu'un HMM est d'abord caractérisé par une

probabilité a_{ij} de passer d'un état à un autre, ensuite qu'à chaque état est associée une probabilité de générer **O** ou **C**. À chaque instant, il y a, non pas un, mais deux tirages aléatoires : le premier pour tirer une lettre de l'alphabet des observations, le second pour changer d'état. L'observation d'une séquence de **O** et de **C** n'est donc plus directement liée à une suite unique d'états. Par exemple, comme on le voit sur la figure 13.3, la séquence **O C C** peut être engendrée avec une certaine probabilité (on verra plus loin comment on la calcule) par la suite d'états **1 2 2** ou la suite **2 2 2**. Dans le modèle présenté, n'importe quelle suite d'états peut en réalité engendrer n'importe quelle suite d'observations avec une certaine probabilité.

Cette différence peut apparaître inutilement subtile. En réalité, elle est très importante. Précisons l'exemple pour mesurer la différence de puissance de modélisation entre un modèle de Markov observable et un HMM. Rappelons que la probabilité pour le modèle de Markov observable d'engendrer une séquence de longueur $2n$ comportant autant de **O** que de **C** est exactement $p^n(1-p)^n$, indépendamment de la répartition des **O** et des **C** dans cette séquence.

Dans le cas du HMM, en prenant a_{11}, p_1, a_{22} et p_2 proches de 1, il est intéressant de constater que la phrase **O O C C** aura une forte probabilité d'être émise, alors que la phrase **C C O O** aura une probabilité faible. Pourtant, ces deux phrases comportent le même nombre de **O** et de **C**. D'une manière générale, une phrase ayant plus de **O** dans sa première moitié aura une probabilité plus forte que sa symétrique d'être émise par ce HMM. Cet exemple peut convaincre que si le HMM est plus complexe que le modèle observable, il a en retour la possibilité de représenter des concepts plus élaborés. En l'occurrence, avec deux états, il est capable de représenter qu'il y a une différence entre les instants d'arrivée des oies et ceux des cygnes⁵. On verra le développement de cet exemple au paragraphe 13.6.

Remarquons aussi ceci : bien que l'alphabet des séquences observables soit composé de deux lettres, le HMM n'a plus de raison d'avoir exactement deux états. La figure 13.4, associée au tableau 13.2, présente un HMM à trois états. Les remarques sur le HMM à deux états sont encore valables : n'importe quelle suite d'états de ce HMM peut engendrer n'importe quelle suite d'observations de **O** et **C** avec une certaine probabilité. Ajoutons la remarque suivante : puisqu'on n'associe pas dans un HMM un état à une observation, il est possible de définir des observations appartenant à un alphabet infini.

Etat	1	2	3
P(O)	p1	p2	p3
P(C)	1-p1	1-p2	1-p3

TAB. 13.2 – *Les probabilités d'émission du HMM à trois états.*

13.2.3 Notations

Un HMM est noté $\Lambda = (A, B, \pi)$ et se définit par :

- Ses états, en nombre n , qui composent l'ensemble $S = \{s_1, s_2, \dots, s_n\}$. L'état où se trouve le HMM à l'instant t est noté q_t ($q_t \in S$).
- M symboles observables dans chaque état. L'ensemble des observations possibles (l'alphabet) est noté $V = \{v_1, v_2, \dots, v_M\}$. $v_t \in V$ est le symbole observé à l'instant t .

5. Pour être complètement exact, un modèle observable pourrait aussi représenter une telle dépendance. Avec deux états, on peut en réalité représenter quatre probabilités différentes pour chaque séquence de deux observations (**O O**, **O C**, etc.) et donc traduire une dépendance d'un événement avec l'événement précédent. En associant cette remarque à celle faite en note de bas de page précédemment, on voit que le pouvoir d'expression des modèles observables peut être augmenté si on les sophistique... mais seulement sur des alphabets finis.

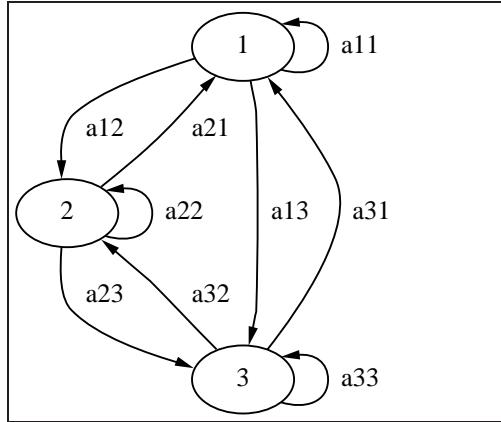


FIG. 13.4 – Le HMM à trois états.

- Une matrice A de *probabilités de transition* entre les états: a_{ij} représente la probabilité que le modèle évolue de l'état i vers l'état j :

$$a_{ij} = A(i, j) = P(q_{t+1} = s_j \mid q_t = s_i) \quad \forall i, j \in [1 \dots n] \quad \forall t \in [1 \dots T]$$

avec :

$$a_{ij} \geq 0 \quad \forall i, j \quad \text{et:} \quad \sum_{j=1}^n a_{ij} = 1$$

- une matrice B de *probabilités d'observation* des symboles dans chacun des états du modèle: $b_j(k)$ représente la probabilité que l'on observe le symbole v_k alors que le modèle se trouve dans l'état j , soit :

$$b_j(k) = P(O_t = v_k \mid q_t = s_j) \quad 1 \leq j \leq n, \quad 1 \leq k \leq M$$

avec :

$$b_j(k) \geq 0 \quad \forall j, k \quad \text{et:} \quad \sum_{k=1}^M b_j(k) = 1$$

- Un vecteur π de *probabilités initiales*: $\pi = \{\pi_i\}_{i=1,2,\dots,n}$. Pour tout état i , π_i est la probabilité que l'état de départ du HMM soit l'état i :

$$\pi_i = P(q_1 = s_i) \quad 1 \leq i \leq n$$

avec :

$$\pi_i \geq 0 \quad \forall i \quad \text{et:} \quad \sum_{i=1}^n \pi_i = 1$$

- Un ou plusieurs *états finals*. Ici, nous supposons pour simplifier que le processus peut s'arrêter dans n'importe quel état, autrement dit que tout état est final.

13.2.4 Deux types de HMM

En pratique, on utilise deux types de modèles de Markov cachés, le modèle *ergodique* et le modèle *gauche-droite*.

Le modèle ergodique est sans contrainte : toutes les transitions d'un état vers un autre sont possibles. Les exemples présentés précédemment sont de ce type.

Le modèle gauche-droite est un modèle contenant des contraintes résultant de la mise à zéro de certaines valeurs a_{ij} . Dans le modèle le plus utilisé, celui de la figure 13.5, l'état i n'est relié par une transition de probabilité non nulle qu'à trois états : lui-même, l'état $i + 1$ et l'état $i - 1$. D'où le nom de *modèle gauche-droite*⁶.

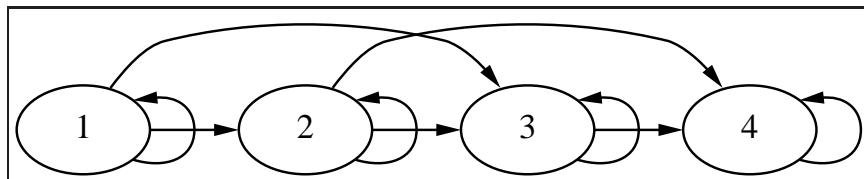


FIG. 13.5 – Le HMM gauche-droite à quatre états.

13.2.5 Comment un HMM engendre une séquence

Un HMM peut être vu comme un processus permettant d'engendrer une séquence ; inversement, on peut considérer une séquence comme une suite d'observations sur un HMM en fonctionnement. En se plaçant du premier point de vue, la génération d'une séquence peut se décrire par l'algorithme 13.1 : c'est une procédure itérative gérée par des tirages aléatoires.

Algorithme 13.1 Génération d'une séquence par un HMM

$t \leftarrow 1$

Choisir l'état initial $q_1 = s_i$ avec la probabilité π_i

tant que $t \leq T$ faire

 Choisir l'observation $o_t = v_k$ avec la probabilité $b_i(k)$

 Passer à l'état suivant $q_{t+1} = s_j$ avec la probabilité a_{ij}

$t \leftarrow t + 1$

fin tant que

Répétons ici qu'une séquence donnée peut en général être engendrée de plusieurs façons distinctes par un HMM.

13.3 Les HMM comme règles de classification de séquences

13.3.1 Les trois problèmes des HMM

Les définitions précédentes ne sont utilisables que si l'on sait calculer la probabilité qu'une séquence soit engendrée par un HMM et surtout si l'on sait apprendre un HMM à partir d'exemples. On doit donc chercher des algorithmes pour résoudre les problèmes suivants :

- *L'évaluation de la probabilité de l'observation d'une séquence.* Etant donnés la séquence d'observations O et un HMM $\Lambda = (A, B, \pi)$, comment évaluer la probabilité d'observation $P(O | \Lambda)$? La réponse à cette question est importante : dans un problème de classification,

6. Ou modèle de Bakis. Dans l'exemple d'introduction, le HMM présenté est encore plus simple.

on attribuera à une séquence la classe que modélise le HMM le plus probable étant donnée la séquence.

- *La recherche du chemin le plus probable.* Étant donnés la suite d'observations O et un HMM Λ , comment trouver une suite d'états $Q = q_1, q_2, \dots, q_T$ qui maximise la probabilité d'observation de la séquence ?
- *L'apprentissage.* Comment ajuster les paramètres (A, B, π) d'un HMM Λ pour maximiser

$$P(O | \Lambda) = \prod_{O \in \mathcal{O}} P(O | \Lambda)$$

à partir d'un ensemble \mathcal{O} de séquences d'apprentissage ?

Notons que la résolution du second problème n'est pas indispensable à l'utilisation des HMM en décision bayésienne. On reviendra sur son utilité au paragraphe 13.7.

13.3.2 Les HMM et la classification bayésienne

Le principe est d'apprendre un HMM par classe à partir des exemples de cette classe. L'apprentissage d'un HMM se fait à partir d'un modèle initial ; le HMM se modifie, mais en gardant jusqu'à sa convergence certaines caractéristiques du modèle initial (une certaine *architecture*) :

- le nombre d'états reste inchangé,
- une transition de probabilité nulle entre deux états du modèle initial garde toujours une valeur nulle.

Le mieux est de prendre pour chaque classe un modèle initial ayant la même architecture : par exemple un modèle ergodique ou un modèle de Bakis. Pour chaque classe, le modèle initial peut simplement être pris avec le même nombre d'états⁷.

Après C apprentissages indépendants, on dispose donc de C HMM, que l'on peut noter $\Lambda(1), \dots, \Lambda(C)$

Étant donnée une séquence quelconque O , on a pour la classe de rang k :

$$P(\Lambda(k) | O) = \frac{P(O | \Lambda(k)).P(\Lambda(k))}{P(O)}$$

Le modèle qui doit être choisi par la règle bayésienne est celui qui maximise $P(\Lambda(k) | O)$ (règle *MAP* : maximum *a posteriori*), ou si l'on suppose les classes équiprobables, celui qui maximise $P(O | \Lambda(k))$ (maximum de vraisemblance), comme indiqué au chapitre 2.

On doit donc être capable de calculer cette dernière valeur pour tout i . Cela nécessite un algorithme capable d'évaluer la probabilité qu'une phrase soit émise par un HMM. C'est le sujet que nous allons développer au paragraphe suivant.

7. Cette simplification n'est nullement nécessaire à l'application du principe *MAP*, mais elle est utilisée en l'absence de connaissances qui pourraient la mettre en cause.

13.4 L'évaluation de la probabilité d'observation

L'évaluation directe

Remarquons d'abord que la probabilité de la suite d'observations O , étant donné le modèle Λ , est égale à la somme sur tous les suites d'états possibles Q des probabilités conjointes de O et de Q :

$$P(O | \Lambda) = \sum_Q P(O, Q | \Lambda) = \sum_Q P(O | Q, \Lambda)P(Q | \Lambda)$$

Or, on a les relations:

$$P(Q | \Lambda) = \pi_{q_1} a_{q_1 q_2} a_{q_2 q_3} \dots a_{q_{T-1} q_T}$$

$$P(O | Q, \Lambda) = b_{q_1}(O_1) b_{q_2}(O_2) \dots b_{q_T}(O_T)$$

On déduit donc des formules précédentes, en réarrangeant les termes :

$$P(O | \Lambda) = \sum_{Q=q_1, q_2, \dots, q_T} \pi_{q_1} b_{q_1}(O_1) a_{q_1 q_2} b_{q_2}(O_2) \dots a_{q_{T-1} q_T} b_{q_T}(O_T)$$

Cette formule directe nécessite d'énumérer toutes les suites d'états de longueur T , soit une complexité en $\Theta(n^T)$. Il existe heureusement une méthode plus rapide.

L'évaluation par les fonctions forward-backward.

Dans cette approche [Bau72], on remarque que l'observation peut se faire en deux temps : d'abord, l'émission du début de l'observation $O(1 : t)$ en aboutissant à l'état q_i au temps t , puis, l'émission de la fin de l'observation $O(t + 1 : T)$ sachant que l'on part de q_i au temps t . Ceci posé, la probabilité de l'observation est donc égale à :

$$P(O | \Lambda) = \sum_{i=1}^n \alpha_t(i) \beta_t(i)$$

où $\alpha_t(i)$ est la probabilité d'émettre le début $O(1 : t)$ et d'aboutir à q_i à l'instant t , et $\beta_t(i)$ est la probabilité d'émettre la fin $O(t + 1 : T)$ sachant que l'on part de q_i à l'instant t . Le calcul de α se fait avec t croissant tandis que le calcul de β se fait avec t décroissant, d'où l'appellation *forward-backward*.

Le calcul de α

On a :

$$\alpha_t(i) = P(O_1 O_2 \dots O_t, q_t = s_i | \Lambda)$$

$\alpha_t(i)$ se calcule par l'algorithme 13.2, qui exprime que pour émettre le début de l'observation $O(1 : t + 1)$ et aboutir dans l'état s_j au temps $t + 1$, on doit nécessairement être dans l'un des états s_i à l'instant t . Cette remarque permet d'exprimer $\alpha_{t+1}(j)$ en fonction des $\alpha_t(i)$ et d'utiliser un algorithme de programmation dynamique pour le calcul de tous les $\alpha_t(i)$ pour tout i , puis des $\alpha_{t+1}(i)$ pour tout i , etc.

Ce calcul a une complexité en $\Theta(n^2 T)$.

Le calcul de β

De manière analogue, $\beta_t(i)$ se calcule par l'algorithme 13.3.

Le calcul de β est lui aussi en $\Theta(n^2 T)$.

Algorithme 13.2 Calcul de la fonction *forward* α

```

pour  $i = 1, n$  faire
     $\alpha_1(i) \leftarrow \pi_i b_i(O_1)$ 
fin pour
 $t \leftarrow 1$ 
tant que  $t < T$  faire
     $j \leftarrow 1$ 
    tant que  $j \leq n$  faire
         $\alpha_{t+1}(j) \leftarrow [\sum_{i=1}^n \alpha_t(i)a_{ij}]b_j(O_{t+1})$ 
         $j \leftarrow j + 1$ 
    fin tant que
     $t \leftarrow t + 1$ 
fin tant que
 $P(O | \Lambda) \leftarrow \sum_{i=1}^n \alpha_T(i)$ 

```

Algorithme 13.3 Calcul de la fonction *backward* β

```

pour  $i = 1, n$  faire
     $\beta_T(i) \leftarrow 1$ 
fin pour
 $t \leftarrow T$ 
tant que  $t > 1$  faire
     $j \leftarrow 1$ 
    tant que  $j \leq n$  faire
         $\beta_t(i) \leftarrow \sum_{j=1}^n a_{ij}b_j(O_{t+1})\beta_{t+1}(j)$ 
         $j \leftarrow j - 1$ 
    fin tant que
     $t \leftarrow t - 1$ 
fin tant que
 $P(O | \Lambda) \leftarrow \sum_{i=1}^n \beta_T(i)$ 

```

Le calcul de la probabilité d'observation

Finalement, la probabilité d'observation d'une séquence est obtenue en prenant les valeurs de α et de β à un instant t quelconque: $P(O | \Lambda) = \sum_{i=1}^n \alpha_t(i)\beta_t(i)$. Cependant, on utilise le plus souvent les valeurs obtenues pour deux cas particuliers ($t = 0$) ou ($t = T$), ce qui donne:

$$P(O | \Lambda) = \sum_{i=1}^n \alpha_T(i) = \sum_{i=1}^n \pi_i \beta_0(i)$$

Exemple

Soit le modèle $\Lambda = (A, B, \pi)$ (figure 13.6) comprenant trois états 1, 2, 3 chacun permettant d'observer un symbole de l'alphabet $V = \{a, b\}$.

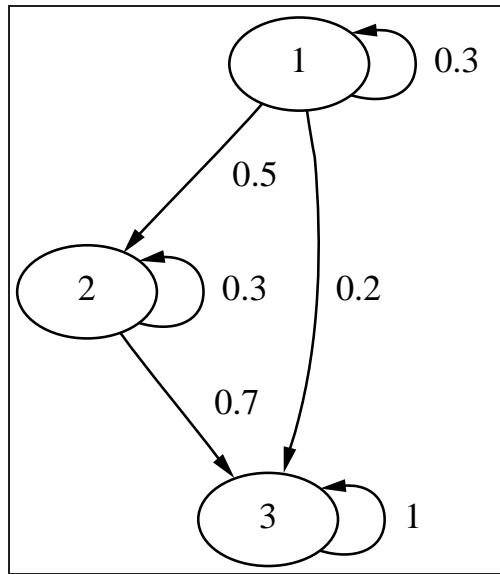


FIG. 13.6 – Un exemple de HMM.

Etat	1	2	3
P(a)	1	0.5	0
P(b)	0	0.5	1

TAB. 13.3 – La matrice B de ce HMM.

$$A = \begin{pmatrix} 0.3 & 0.5 & 0.2 \\ 0 & 0.3 & 0.7 \\ 0 & 0 & 1 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 0 \\ 0.5 & 0.5 \\ 0 & 1 \end{pmatrix} \quad \pi = \begin{pmatrix} 0.6 \\ 0.4 \\ 0 \end{pmatrix}$$

La figure 13.7 illustre le calcul de α pour la suite d'observations : $a\ a\ b\ b$.

$$\begin{aligned}
 \alpha_1(1) &= \pi_1 b_1(a) = 0.6 \times 1 = 0.6 \\
 \alpha_1(2) &= \pi_2 b_2(a) = 0.4 \times 0.5 = 0.2 \\
 \alpha_1(3) &= \pi_3 b_3(a) = 0 \times 0 = 0 \\
 \alpha_2(1) &= (\alpha_1(1)a_{11} + \alpha_1(2)a_{21} + \alpha_1(3)a_{31})b_1(a) \\
 &= (0.6 \times 0.3 + 0.2 \times 0 + 0 \times 0) \times 1 \\
 &= (0.18) \times 1 = 0.18 \\
 \alpha_2(2) &= (\alpha_1(1)a_{12} + \alpha_1(2)a_{22} + \alpha_1(3)a_{32})b_2(a) \\
 &= (0.6 \times 0.5 + 0.2 \times 0.3 + 0 \times 0) \times 0.5 \\
 &= (0.36) \times 0.5 = 0.18 \\
 \dots &\quad \dots \\
 P(a\ a\ b\ b \mid \Lambda) &= \sum_{q_i} \alpha_4(i) = 0.2228
 \end{aligned}$$

13.5 Le calcul du chemin optimal : l'algorithme de Viterbi

Il s'agit maintenant de déterminer le meilleur chemin correspondant à l'observation, c'est-à-dire de trouver dans le modèle Λ la *meilleure suite d'états* Q , qui maximise la quantité :

$$P(Q, O \mid \Lambda)$$

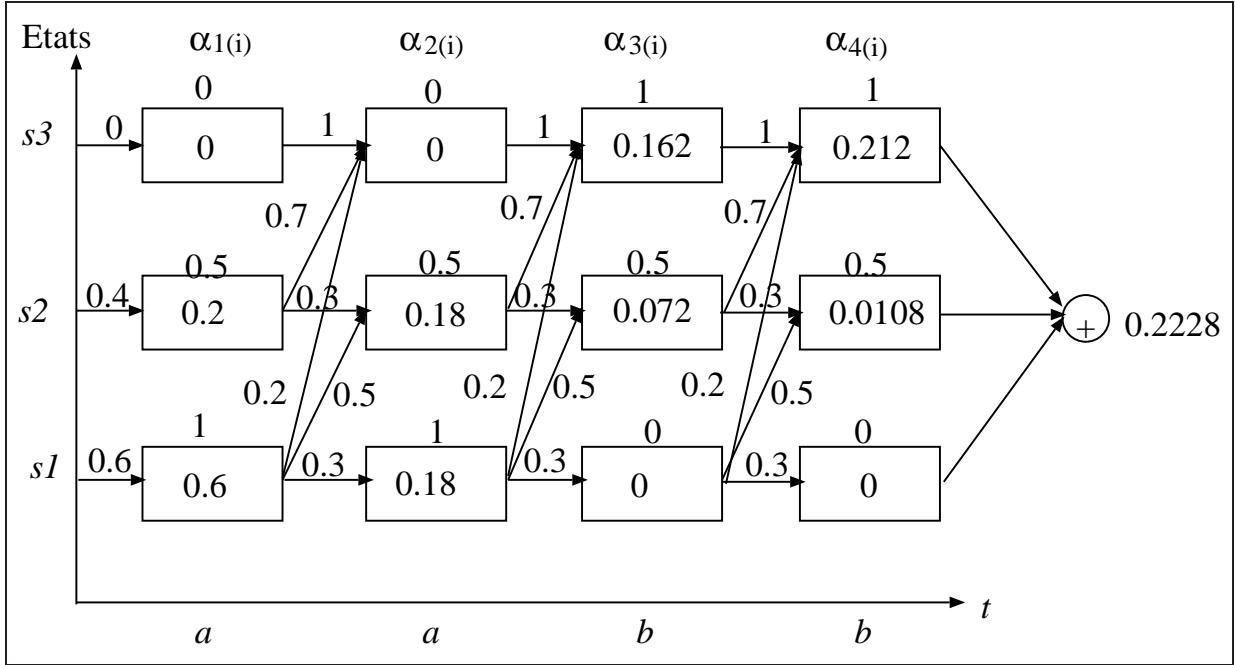


FIG. 13.7 – Calcul de α pour la suite d’observations “aabb”.

Pour trouver $Q = (q_1, q_2, \dots, q_T)$ pour une séquence d’observations $O = (O_1, O_2, \dots, O_T)$, on définit la variable intermédiaire $\delta_t(i)$ comme la probabilité du meilleur chemin amenant à l’état s_i à l’instant t , en étant guidé par les t premières observations :

$$\delta_t(i) = \underset{q_1, \dots, q_{t-1}}{\text{Max}} P(q_1, q_2, \dots, q_t = s_i, O_1, O_2, \dots, O_t | \Lambda)$$

Par récurrence, on calcule

$$\delta_{t+1}(j) = [\underset{i}{\text{Max}} \delta_t(i) a_{ij}] b_j(O_{t+1})$$

en gardant trace, lors du calcul, de la suite d’états qui donne le meilleur chemin amenant à l’état s_i à t dans un tableau ψ .

On utilise une variante de la programmation dynamique, l'*algorithme de Viterbi* (algorithme 13.4) pour formaliser cette récurrence. Il fournit en sortie la valeur P^* de la probabilité de l’émission de la séquence par la meilleure suite d’états (q_1^*, \dots, q_T^*)

La fonction *Argmax* permet de mémoriser l’indice i , entre 1 et n , avec lequel on atteint le maximum des quantités $(\delta_{t-1}(i) a_{ij})$. Le coût des opérations est également en $\Theta(n^2T)$.

Exemple [BB92]

À partir de la figure 13.8 qui illustre le calcul de δ , on peut calculer les quantités δ , ψ et q^* comme suit :

Algorithme 13.4 Algorithme de Viterbi

```

pour  $i = 1, n$  faire
     $\delta_1(i) \leftarrow \pi_i b_i(O_1)$ 
     $\psi_1(i) \leftarrow 0$ 
fin pour
 $t \leftarrow 2$ 
tant que  $t \leq T$  faire
     $j \leftarrow 1$ 
    tant que  $j \leq n$  faire
         $\delta_t(j) \leftarrow \underset{1 \leq i \leq n}{\text{Max}} [\delta_{t-1}(i)a_{ij}]b_j(O_t)$ 
         $\psi_t(j) \leftarrow \underset{1 \leq i \leq n}{\text{ArgMax}} [\delta_{t-1}(i)a_{ij}]$ 
         $j \leftarrow j + 1$ 
    fin tant que
     $t \leftarrow t + 1$ 
fin tant que
 $P^* \leftarrow \underset{1 \leq i \leq n}{\text{Max}} [\delta_T(i)]$ 
 $q_T^* \leftarrow \underset{1 \leq i \leq n}{\text{ArgMax}} [\delta_T(i)]$ 
 $t \leftarrow T$ 
tant que  $t \geq 1$  faire
     $q_t^* \leftarrow \psi_{t+1}(q_{t+1}^*)$ 
     $t \leftarrow t - 1$ 
fin tant que

```

$$\begin{aligned}
 \delta_1(1) &= \pi_1 b_1(a) = 0.6 \times 1 = 0.6 & \psi_1(1) &= 0, \\
 \delta_1(2) &= \pi_2 b_2(a) = 0.4 \times 0.5 = 0.2 & \psi_1(2) &= 0, \\
 \delta_1(3) &= \pi_3 b_3(a) = 0 \times 0 = 0 & \psi_1(3) &= 0, \\
 \delta_2(1) &= \max_{1 \leq i \leq n} (\delta_1(i)a_{i1})b_1(a)
 \end{aligned}$$

$$\begin{aligned}
 &= \max \left\{ \begin{array}{l} \delta_1(1)a_{11} \\ \delta_1(2)a_{21} \\ \delta_1(3)a_{31} \end{array} \right\} \times b_1(a) \\
 &= \max \left\{ \begin{array}{l} 0.6 \times 0.3 \\ 0 \\ 0 \end{array} \right\} \times 1 = 0.18 \quad \psi_2(1) = 1,
 \end{aligned}$$

...

Finalement :

$$\begin{array}{llll}
 \underline{\psi_1(1) = 0} & \underline{\psi_2(1) = 1} & \underline{\psi_3(1) = 1} & \underline{\psi_4(1) = 1} \\
 \underline{\psi_1(2) = 0} & \underline{\psi_2(2) = 1} & \underline{\psi_3(2) = 1} & \underline{\psi_4(2) = 2} \\
 \underline{\psi_1(3) = 0} & \underline{\psi_2(3) = 2} & \underline{\psi_3(3) = 2} & \underline{\psi_4(3) = 3},
 \end{array}$$

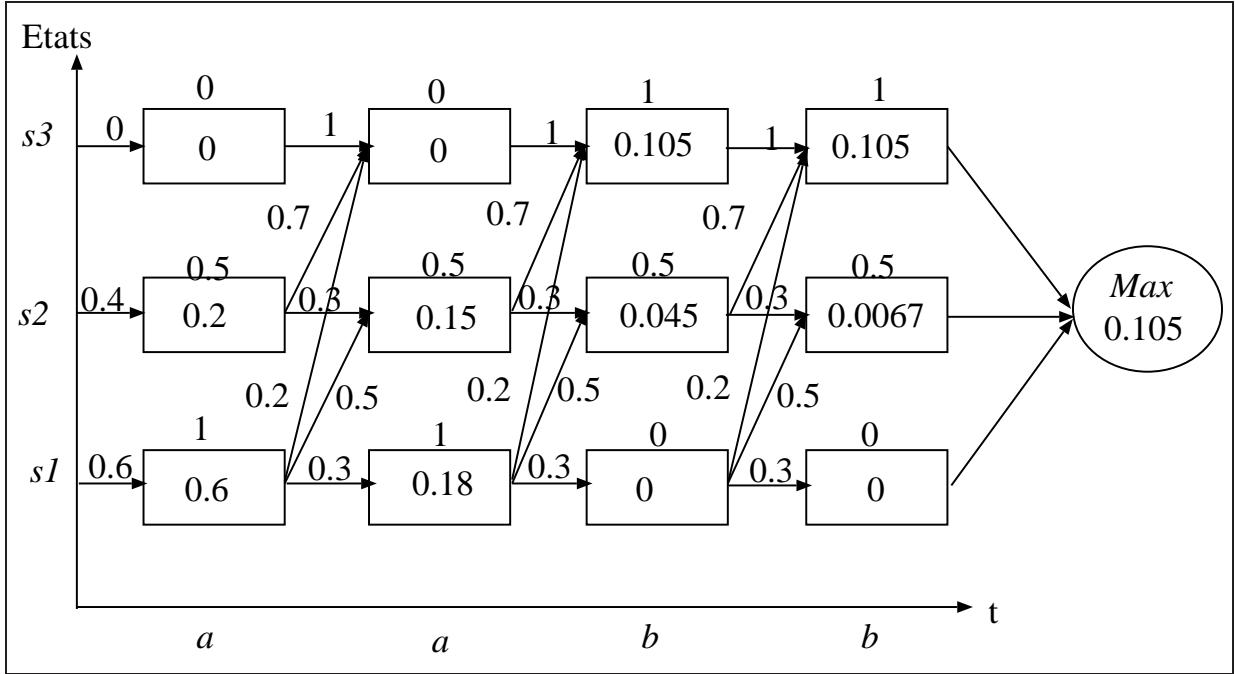


FIG. 13.8 – Calcul de δ pour la suite d’observations “aabb”.

$$q_4^* = \max \begin{pmatrix} \delta_4(1) \\ \delta_4(2) \\ \delta_4(3) \end{pmatrix} = 3,$$

$$\begin{aligned} q_3^* &= \psi_4(3) &= 3, \\ q_2^* &= \psi_3(3) &= 2, \\ q_1^* &= \psi_2(2) &= 1. \end{aligned}$$

On déduit donc de ce calcul que la *meilleure suite d’états*, celle qui engendre la phrase $a\ a\ b\ b$ avec la plus forte probabilité, est : 1 2 3 3.

13.6 L’apprentissage

Principe

Supposons disposer d’un ensemble de séquences $\mathcal{O} = \{O^1, \dots, O^m\}$, dont l’élément courant est noté O^k . Le but de l’apprentissage est de déterminer les paramètres d’un HMM d’architecture fixée : $\Lambda = (A, B, \pi)$, qui maximisent la probabilité $P(\mathcal{O} | \Lambda)$. Comme on suppose les séquences d’apprentissages tirées indépendamment, on cherche donc à maximiser :

$$P(\mathcal{O} | \Lambda) = \prod_{k=1}^m P(O^k | \Lambda)$$

L’idée est d’utiliser une procédure de réestimation qui affine le modèle petit à petit selon les étapes suivantes :

- choisir un ensemble initial Λ_0 de paramètres ;

- calculer Λ_1 à partir de Λ_0 , puis Λ_2 à partir de Λ_1 , etc.
- répéter ce processus jusqu'à un critère de fin.

Pour chaque étape p d'apprentissage, on dispose de Λ_p et on cherche un Λ_{p+1} qui doit vérifier :

$$P(\mathcal{O} \mid \Lambda_{p+1}) \geq P(\mathcal{O} \mid \Lambda_p)$$

soit :

$$\prod_{k=1}^m P(O^k \mid \Lambda_{p+1}) \geq \prod_{k=1}^m P(O^k \mid \Lambda_p)$$

Λ_{p+1} doit donc améliorer la probabilité de l'émission des observations de l'ensemble d'apprentissage. La technique pour calculer Λ_{p+1} à partir de Λ_p consiste à utiliser l'algorithme *EM*. Pour cela, on fait un comptage de l'utilisation des transitions A et des distributions B et π du modèle Λ_p quand il produit l'ensemble \mathcal{O} . Si cet ensemble est assez important, ces fréquences fournissent de bonnes approximations *a posteriori* des distributions de probabilités A, B et π et sont utilisables alors comme paramètres du modèle Λ_{p+1} pour l'itération suivante.

La méthode d'apprentissage *EM* consiste donc dans ce cas à regarder comment se comporte le modèle défini par Λ_p sur \mathcal{O} , à réestimer ses paramètres à partir des mesures prises sur \mathcal{O} , puis à recommencer cette réestimation jusqu'à obtenir une convergence. L'annexe 18.9 donne quelques détails sur cette méthode.

Dans les calculs qui suivent, on verra apparaître en indice supérieur la lettre k quand il faudra faire référence à la séquence d'apprentissage concernée. L'indice p , qui compte les passes d'apprentissage, sera omis : on partira d'un modèle noté simplement Λ et on calculera celui qui s'en déduit.

Les formules de réestimation

On définit $\xi_t^k(i, j)$ comme la probabilité, étant donnés une phrase O^k et un HMM Λ , que ce soit l'état s_i qui ait émis la lettre de rang t de O^k et l'état s_j qui ait émis celle de rang $t + 1$. Donc :

$$\xi_t^k(i, j) = P(q_t = s_i, q_{t+1} = s_j \mid O^k, \Lambda)$$

Ce qui se récrit :

$$\xi_t^k(i, j) = \frac{P(q_t = s_i, q_{t+1} = s_j, O^k \mid \Lambda)}{P(O^k \mid \Lambda)}$$

Par définition des fonctions *forward-backward*, on en déduit :

$$\xi_t^k(i, j) = \frac{\alpha_t^k(i) a_{ij} b_j(O_{t+1}^k) \beta_{t+1}^k(j)}{P(O^k \mid \Lambda)}$$

On définit aussi la quantité $\gamma_t^k(i)$ comme la probabilité que la lettre de rang t de la phrase O^k soit émise par l'état s_j .

$$\gamma_t^k(i) = P(q_t = s_i \mid O^k, \Lambda)$$

Soit :

$$\gamma_t^k(i) = \sum_{j=1}^n P(q_t = s_i, q_{t+1} = s_j \mid O^k, \Lambda) = \frac{\sum_{j=1}^n P(q_t = s_i, q_{t+1} = s_j, O^k \mid \Lambda)}{P(O^k \mid \Lambda)}$$

On a la relation :

$$\gamma_t^k(i) = \sum_{j=1}^n \xi_t^k(i, j) = \frac{\alpha_t^k(i) \beta_t^k(i)}{P(O^k \mid \Lambda)}$$

Le nouveau modèle HMM se calcule à partir de l'ancien en réestimant π , A et B par comptage sur la base d'apprentissage. On mesure les fréquences :

$$\begin{aligned}\bar{a}_{ij} &= \frac{\text{nombre de fois où la transition de } s_i \text{ à } s_j \text{ a été utilisée}}{\text{nombre de transitions effectuées à partir de } s_i} \\ \bar{b}_j(l) &= \frac{\text{nombre de fois où le HMM s'est trouvé dans l'état } s_j \text{ en observant } v_l}{\text{nombre de fois où le HMM s'est trouvé dans l'état } s_j} \\ \bar{\pi}_i &= \frac{\text{nombre de fois où le HMM s'est trouvé dans l'état } s_i \dots}{\text{nombre de fois où le HMM ...}} \\ &\quad \dots \text{ en émettant le premier symbole d'une phrase} \\ &\quad \dots \text{ a émis le premier symbole d'une phrase}\end{aligned}$$

Compte tenu de ces définitions :

$$\begin{aligned}\bar{\pi}_i &= \frac{1}{m} \sum_{k=1}^m \gamma_1^k(i) \\ \bar{a}_{ij} &= \frac{\sum_{k=1}^m \sum_{t=1}^{|O^k|-1} \xi_t^k(i, j)}{\sum_{k=1}^N \sum_{t=1}^{|O^k|-1} \gamma_t^k(i)} \\ \bar{b}_j(l) &= \frac{\sum_{k=1}^m \sum_{\substack{t=1 \\ \text{avec } O_t^k=v_l}}^{|O^k|-1} \gamma_t^k(j)}{\sum_{k=1}^m \sum_{t=1}^{|O^k|-1} \gamma_t^k(j)}\end{aligned}$$

Ces formules ont été établies par Baum ([Bau72]), comme une application de la procédure *EM* à l'apprentissage des paramètres HMM. La suite des modèles construits par l'algorithme de Baum-Welsh ([RJ93]) vérifie la relation cherchée :

$$P(\mathcal{O} \mid \Lambda_{p+1}) \geq P(\mathcal{O} \mid \Lambda_p)$$

Remarques :

- Le choix du modèle initial influe sur les résultats ; par exemple, si certaines valeurs de A et B sont égales à 0 au départ, elles le resteront jusqu'à la fin de l'apprentissage. Ceci permet en particulier de garder la structure dans les modèles gauches-droits.
- L'algorithme converge vers des valeurs de paramètres qui assurent un *maximum local* de $P(O \mid \Lambda)$. Il est donc important, si l'on veut être aussi près que possible du minimum global, de bien choisir la structure et l'initialisation.
- Le nombre d'itérations est fixé empiriquement. L'expérience prouve que, si le point précédent a été correctement traité, la stabilisation des paramètres ne correspond pas à un surapprentissage : il n'y a donc en général pas besoin de contrôler la convergence par un ensemble de validation. Mais cette possibilité est évidemment toujours à disposition.

Algorithme 13.5 Algorithme de Baum-Welch

Fixer des valeurs initiales (A, B, π)

On définit le HMM de départ comme $\Lambda_0 = (A, B, \pi)$.

$p \leftarrow 0$

tant que la convergence n'est pas réalisée **faire**

On possède le HMM Λ_p .

On calcule pour ce modèle, sur l'ensemble d'apprentissage, les valeurs :

$$\xi(i, j), \quad \gamma_t(i) \quad 1 \leq i, j \leq n \quad 1 \leq t \leq T - 1$$

On en déduit $\bar{\pi}$, \bar{A} , \bar{B} en utilisant les formules de réestimation.

Le HMM courant est désormais défini par $\Lambda_p = (\bar{\pi}, \bar{A}, \bar{B})$

$p \leftarrow p + 1$

fin tant que

Un exemple

En partant du HMM Λ_0 défini par les paramètres suivants :

$$A = \begin{pmatrix} 0.45 & 0.35 & 0.20 \\ 0.10 & 0.50 & 0.40 \\ 0.15 & 0.25 & 0.60 \end{pmatrix} \quad B = \begin{pmatrix} 1.0 & 0.0 \\ 0.5 & 0.5 \\ 0.0 & 1.0 \end{pmatrix} \quad \pi = \begin{pmatrix} 0.5 \\ 0.3 \\ 0.2 \end{pmatrix}$$

on peut calculer que, s'il émet sur l'alphabet à deux lettres $V = \{a, b\}$, on a :

$$P(a \ b \ b \ a \ a \mid \Lambda_0) = 0.0278$$

Si on prend comme ensemble d'apprentissage cette seule phrase, l'application de l'algorithme de Baum-Welch doit augmenter sa probabilité de reconnaissance.

Après une réestimation⁸, on trouve le HMM Λ_1 :

$$A = \begin{pmatrix} 0.346 & 0.365 & 0.289 \\ 0.159 & 0.514 & 0.327 \\ 0.377 & 0.259 & 0.364 \end{pmatrix} \quad B = \begin{pmatrix} 1.0 & 0.0 \\ 0.631 & 0.369 \\ 0.0 & 1.0 \end{pmatrix} \quad \pi = \begin{pmatrix} 0.656 \\ 0.344 \\ 0.0 \end{pmatrix}$$

$$P(a \ b \ b \ a \ a \mid \Lambda_1) = 0.0529$$

Après quinze itérations :

$$A = \begin{pmatrix} 0.0 & 0.0 & 1.0 \\ 0.212 & 0.788 & 0.0 \\ 0.0 & 0.515 & 0.485 \end{pmatrix} \quad B = \begin{pmatrix} 1.0 & 0.0 \\ 0.969 & 0.031 \\ 0.0 & 1.0 \end{pmatrix} \quad \pi = \begin{pmatrix} 1.0 \\ 0.0 \\ 0.0 \end{pmatrix}$$

$$P(a \ b \ b \ a \ a \mid \Lambda_{15}) = 0.2474$$

Après cent cinquante itérations, la convergence est réalisée. La Figure 13.9 et le tableau 13.4 décrivent le résultat, que l'on peut donner aussi sous la forme suivante :

8. Les calculs sont très pénibles à faire à la main, même sur un exemple simple comme celui-ci.

$$A = \begin{pmatrix} 0.0 & 0.0 & 1.0 \\ 0.18 & 0.82 & 0.0 \\ 0.0 & 0.5 & 0.5 \end{pmatrix} \quad B = \begin{pmatrix} 1.0 & 0.0 \\ 1.0 & 0.0 \\ 0.0 & 1.0 \end{pmatrix} \quad \pi = \begin{pmatrix} 1.0 \\ 0.0 \\ 0.0 \end{pmatrix}$$

$$P(a \ b \ b \ a \ a \mid \Lambda_{150}) = 0.2500$$

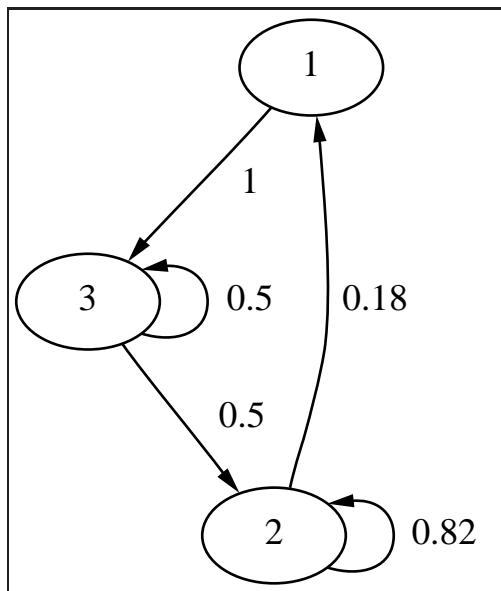


FIG. 13.9 – le HMM entraîné sur une seule phrase, après convergence. Le seul état initial possible est l'état 1.

Etat	1	2	3
P(a)	1	1	0
P(b)	0	0	1

TAB. 13.4 – La matrice B de ce HMM.

Il peut paraître curieux que ce HMM ne génère pas son unique phrase d'apprentissage avec la probabilité 1 et toutes les autres séquences avec la probabilité 0. Ceci vient du fait que le nombre des états est trop petit pour réaliser un apprentissage par coeur. Mais si l'on part d'un HMM initial à cinq états, il converge en revanche vers un HMM Λ défini par :

$$A = \begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix} \quad B = \begin{pmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \\ 0.0 & 1.0 \\ 1.0 & 0.0 \\ 1.0 & 0.0 \end{pmatrix} \quad \pi = \begin{pmatrix} 1.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \end{pmatrix}$$

et l'on a :

$$P(a \ b \ b \ a \ a \mid \Lambda) = 1.0$$

Ce HMM est en réalité une variante d'un modèle observable : à chaque état est associée l'émission certaine d'une lettre. La différence avec les modèles présentés en début de ce chapitre est que la même lettre peut être associée à plusieurs états. Ici, par exemple, la lettre a aux états 1, 4 et 5.

Un autre exemple

Reprenons maintenant l'exemple du paragraphe 13.2.2. Nous partons du HMM suivant, dont les paramètres ont été tirés aléatoirement :

$$A = \begin{pmatrix} 0.40 & 0.60 \\ 0.52 & 0.48 \end{pmatrix} \quad B = \begin{pmatrix} 0.49 & 0.51 \\ 0.40 & 0.60 \end{pmatrix} \quad \pi = \begin{pmatrix} 0.31 \\ 0.69 \end{pmatrix}$$

Nous lui faisons subir deux apprentissages sur deux ensembles de séquences : le premier est composé d'éléments ayant à peu près autant de a que de b , ces derniers étant situés en majorité dans la seconde partie ; l'autre ensemble d'apprentissage est composé de phrases de type symétrique.

$$\mathcal{O}_1 = \{aaabb, abaabbb, aaababb, aabab, ab\}$$

$$\mathcal{O}_2 = \{bbbba, babba, bbbabaa, bbabba, bbaa\}$$

Après convergence, on obtient deux HMM différents donnés sur la figure 13.10.

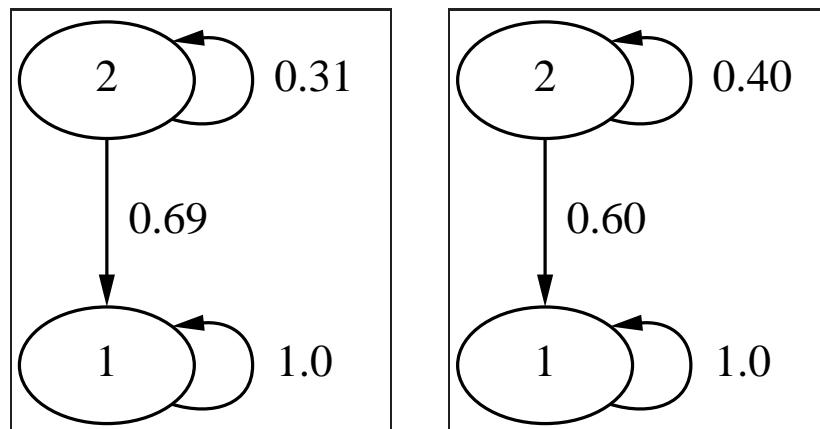


FIG. 13.10 – les HMM $\Lambda(1)$ et $\Lambda(2)$.

$\Lambda(1)$ est défini par :

$$A = \begin{pmatrix} 1.0 & 0.0 \\ 0.69 & 0.31 \end{pmatrix} \quad B = \begin{pmatrix} 0.36 & 0.64 \\ 1.0 & 0.0 \end{pmatrix} \quad \pi = \begin{pmatrix} 0.0 \\ 1.0 \end{pmatrix}$$

Son unique état de départ est l'état 2 qui émet a avec la probabilité 1. La probabilité d'émettre b par l'état 1 est de 0.64.

$\Lambda(2)$ est défini par :

$$A = \begin{pmatrix} 1.0 & 0.0 \\ 0.60 & 0.40 \end{pmatrix} \quad B = \begin{pmatrix} 0.65 & 0.34 \\ 0.0 & 1.0 \end{pmatrix} \quad \pi = \begin{pmatrix} 0.0 \\ 1.0 \end{pmatrix}$$

Son unique état de départ est l'état 2 qui émet b avec la probabilité 1. La probabilité d'émettre a par l'état 1 est de 0.65.

Les deux HMM appris sont donc assez semblables, quand on permute a et b . On remarque qu'ils ne font pas jouer un rôle symétrique à leurs états.

Sur deux phrases n'ayant pas participé à l'apprentissage, on obtient le résultat attendu :

$$P(a \ a \ b \ a \ b \ b \ | \ \Lambda(1)) = 0.0437$$

$$P(a \ a \ b \ a \ b \ b \ | \ \Lambda(2)) = 0.000$$

$$P(b \ b \ a \ b \ a \ a \ | \ \Lambda(1)) = 0.000$$

$$P(b \ b \ a \ b \ a \ a \ | \ \Lambda(2)) = 0.0434$$

13.7 Approfondissements

Comme on l'a dit plus haut, il est possible de définir des HMM produisant des séquences de valeurs continues et même des séquences de vecteurs de valeurs continues. Dans ce cas, elles ne sont évidemment plus construites sur un alphabet fini. Il faut alors remplacer la matrice B par un ensemble de distributions éventuellement multidimensionnelles de probabilités ; les calculs précédents restent valables, mais les formules, en particulier celles de la réestimation pour l'apprentissage doivent porter sur des paramètres caractéristiques des distributions de probabilité en question. Généralement, on suppose celles-ci gaussiennes, ou multigaussiennes.

C'est souvent le cas en reconnaissance de la parole: prenons l'exemple de la reconnaissance du vocabulaire des dix chiffres. L'échantillon d'apprentissage permet de créer dix ensembles d'exemples, chacun supervisé par un chiffre différent. Pour chaque chiffre, on va apprendre un HMM. En phase de reconnaissance, un son inconnu sera classé comme le chiffre associé au HMM qui peut l'émettre avec la plus forte probabilité. Qu'est-ce qu'une séquence représentant la prononciation d'un son ? Au départ, un signal échantillonné, soit 8 ou 16 K valeurs réelles par seconde. Ce signal est transformé par des techniques de type transformation de Fourier pour en extraire ses caractéristiques fréquentielles. Au final, on dispose en général d'un codage de chaque centième de seconde de parole émise par un vecteur d'une dizaine de valeurs réelles. La prononciation d'un chiffre est donc une séquence dont la longueur est de l'ordre de quelques dizaines et dont chaque élément est un vecteur de \mathbb{R}^{10} .

Il est dès lors impossible de définir la matrice B puisque dans chaque terme $b_j(k)$, k devrait parcourir tous les vecteurs différents représentant un centième de seconde de parole et donc prendre un nombre infini de valeurs. Le plus simple est de faire une estimation paramétrique de la distribution des composantes du vecteur émis par chaque état, comme au chapitre 14. On supposera par exemple que ce sont des tirages aléatoires d'une gaussienne de dimension 10: dans ce cas, pour chaque état, il faudra estimer la moyenne et la covariance de la densité de probabilité d'émission d'un vecteur de dimension 10. L'algorithme de Baum-Welsh s'adapte facilement à cette situation.

Des approfondissements et des alternatives ont été proposés: il est commun de supposer que les vecteurs émis sont des *sommes pondérées* de plusieurs distributions gaussiennes multidimensionnelles. L'algorithme *EM* permet encore de calculer les moyennes et les covariances de chacune, ainsi que les coefficients de pondération (voir le chapitre 15). Il a été aussi proposé d'estimer les densités d'émission par la méthode non-paramétrique des K plus proches voisins qui est présenté au chapitre 14 ou par des réseaux connexionnistes (chapitre 10).

13.8 Applications

Les HMM sont actuellement les outils d'apprentissage les plus efficaces pour la classification des séquences : ils ne réclament que peu de connaissances *a priori*; à condition de disposer de suffisamment de données d'apprentissage, ils sont très efficaces. Un grand nombre de raffinements leur ont été apportés, en particulier pour résoudre des problèmes aussi complexes que celui de la reconnaissance de la parole ou de l'écriture manuscrite. À l'heure actuelle, par exemple, presque tous les systèmes de reconnaissance de la parole sont construits à base de HMM, parfois hybrides de réseaux connexionnistes.

Dans la plupart des cas, les informations phonétiques, lexicales, syntaxiques sont « compilées » dans un HMM de plusieurs centaines d'états, dont chacun vise à posséder une signification linguistique; l'apprentissage se fait sur des très grandes quantités de données. La reconnaissance est effectuée en récupérant la séquence d'états par l'algorithme de Viterbi. En effet, les états ont dans ce cas une signification linguistique. Ce n'est pas tant la probabilité finale qui est intéressante que le chemin par lequel passe la meilleure façon d'engendrer le signal.

Notes historiques et sources bibliographiques

L'algorithme de Baum-Welsh ([Bau72]) est une application aux modèles de Markov de la technique générale *EM* d'estimation statistique de paramètres cachés. L'annexe 18.9 qui traite de cette technique est en partie reprise du document [Ros97]. L'utilisation en apprentissage vient de la communauté de la reconnaissance de la parole. Les premières références que l'on peut relier à cette technique datent des années 1970 ([Jel76, Bak75]). L'utilisation en reconnaissance bayésienne pour des séquences, comme présentée dans ce chapitre, est surtout fondée sur les travaux de Rabiner et de son équipe, qui ont démarré vers 1983 ([LRS83]). L'article [Rab89] est un exposé incontournable sur les HMM et leur application à la parole. Les ouvrages [Jel97] et surtout [RJ93] donnent un état de l'art de l'utilisation des HMM en reconnaissance de la parole. Pour leurs applications au traitement des séquences biologiques et des images, deux références récentes sont [BB01] et [BS97, MS01]. L'exemple 13.6 et les figures associées sont empruntés, avec l'aimable accord des auteurs, à [BB92]. Ce dernier ouvrage (entre autres mérites) présente une bonne introduction à ces techniques d'apprentissage et à leur application pour la reconnaissance de l'écriture manuscrite; ce chapitre lui a fait quelques autres emprunts.

Résumé

- les HMM sont des modèles probabilistes d'émission de séquences, discrètes ou continues (et dans ce cas, éventuellement vectorielles)
- en version de base, ils sont utilisés en classification bayésienne
- l'algorithme *forward-backward* permet de connaître la probabilité qu'un HMM ait émis une séquence
- l'algorithme de *Viterbi* permet de connaître la suite des états du HMM qui a la plus forte probabilité d'avoir émis une séquence
- l'algorithme de *Baum-Welsh* permet d'ajuster les paramètres d'un HMM au maximum de vraisemblance à partir d'un ensemble de séquences d'apprentissage

Quatrième partie

**Apprentissage par approximation et
interpolation**

Chapitre 14

L'apprentissage bayésien et son approximation

Nous savons depuis le chapitre 2 que l'apprentissage bayésien consiste à partir d'hypothèses a priori pour les réviser en fonction des données d'apprentissage. Cette opération est optimale au sens probabiliste : les hypothèses a posteriori obtenues de la sorte sont en effet les plus vraisemblables.

D'un point de vue opérationnel, l'application de l'apprentissage bayésien nécessite donc, en principe, d'une part une connaissance a priori sur la vraisemblance des hypothèses en concurrence, mais d'autre part celle la probabilité des données d'apprentissage conditionnellement à ces hypothèses. Si ces valeurs sont connues, l'hypothèse la plus vraisemblable compte tenu des données pourra être sélectionnée. En pratique, ces valeurs ne sont pas connues exactement : elles doivent être estimées. Le but de ce chapitre est donc de présenter quelques méthodes permettant de réaliser ces estimations.

Il existe une littérature très abondante sur le sujet, provenant des statistiques et de la reconnaissance des formes ; cette dernière discipline insiste sur le côté algorithmique de l'estimation. Nous nous plaçons également de ce côté. Nous présentons d'abord les bases de l'apprentissage bayésien, en particulier dans le cas de la classification, et nous décrivons rapidement les liens entre cette approche et d'autres concepts utiles pour l'apprentissage automatique comme la régression, la comparaison de distributions de probabilités, le principe MDL. Nous décrivons ensuite les méthodes paramétriques d'estimation, qui connaissent un regain de faveur en raison de leurs liens avec les fonctions noyau, elles-mêmes liées aux séparateurs à vaste marge (chapitre 9) et aux réseaux connexionnistes (chapitre 10). Les méthodes non paramétriques, ou K plus proches voisins, sont ensuite présentées, avec une insistance sur l'aspect algorithmique. Pour finir, les méthodes semi paramétriques sont abordées et le lien est fait avec d'autres méthodes d'apprentissage, comme les arbres de décision (chapitre 11).

VOICI L'HISTOIRE d'un étudiant en mathématiques qui décide pour se changer les idées de s'initier à l'ornithologie, l'art et la science d'identifier les oiseaux. Il se rend vers un étang sur lequel il a déjà aperçu nager des gros oiseaux. Il y en a quelques-uns ce jour-là. Mais comment savoir à quelle espèce ils appartiennent ? Non loin de là se trouve un autre observateur équipé d'une longue-vue. « Pardon, dit l'étudiant, savez vous quel genre d'oiseaux on voit ici ? » « Certainement, répond l'autre. Il n'y a jamais que des cygnes et des oies. Et depuis des années que je viens ici, j'ai eu l'occasion de compter au total environ trois fois plus de cygnes que d'oies. » « Très bien, merci », dit l'étudiant, avant de se livrer à une observation plus attentive.

Il remarque vite que sans une paire de jumelles, tout ce qu'il peut voir sont des oiseaux plutôt blancs et d'autres plutôt sombres. « Excusez-moi de vous interrompre à nouveau, dit-il à son voisin, pouvez-vous m'indiquer le nom des oiseaux blancs et celui des oiseaux noirs ? » L'expert, un peu agacé, lui répond que c'est plus compliqué que cela : « Il y a des cygnes sombres : les jeunes de moins d'un an. Les adultes sont blancs. La durée de vie d'un cygne est d'environ vingt ans, soit dit en passant. D'autre part, les oies sauvages sont sombres, mais des oies blanches de la ferme d'à côté viennent parfois jusqu'ici. Je dirais qu'une oie sur dix est blanche. Est-ce que cela vous suffit ? » « Parfaitement, merci ». Avisant un oiseau sombre, l'étudiant fait un petit calcul puis montre le bestiau à l'expert. « Je te prend le pari que nage ici une oie ». L'autre, à la longue-vue, regarde, voit, s'étonne. « Il s'agit bien d'une oie. Tu n'es point débutant ! ». « Je le suis. Mais chez moi la raison aide l'observation. »

Que sait l'étudiant ? Que neuf oies sur dix sont sombres, qu'un cygne sur vingt est sombre et qu'un oiseau a trois chances sur quatre d'être un cygne. Que peut-il déduire rationnellement quand il observe un oiseau sombre ?

Avant toute observation, la probabilité *a priori* pour un oiseau d'être un cygne vaut $3/4$ et elle vaut $1/4$ d'être une oie. La probabilité qu'un oiseau soit sombre sachant qu'il est un cygne vaut $1/20$, celle qu'il soit clair sachant qu'il est un cygne vaut $19/20$. La probabilité qu'un oiseau soit sombre sachant qu'il est une oie vaut $9/10$, celle qu'il soit clair sachant qu'il est une oie vaut $1/10$.

De manière plus formelle :

- $P(\text{oiseau} = \text{cygne}) = 3/4$. Notons en raccourci : $P(\text{cygne}) = 3/4$
- $P(\text{oie}) = 1/4$
- $P(\text{couleur} = \text{sombre} \mid \text{oiseau} = \text{cygne}) = P(\text{sombre} \mid \text{cygne}) = 1/20$.
- $P(\text{clair} \mid \text{cygne}) = 19/20$
- $P(\text{sombre} \mid \text{oie}) = 9/10$
- $P(\text{clair} \mid \text{oie}) = 1/10$

Comme l'oiseau observé est sombre, il faut calculer la probabilité conditionnelle $P(\text{oie} \mid \text{sombre})$ qu'il soit une oie sachant que sa couleur est sombre et la probabilité conditionnelle $P(\text{cygne} \mid \text{sombre})$ qu'il soit un cygne sachant que sa couleur est sombre. Il faut ensuite comparer ces deux probabilités conditionnelles. La plus forte des deux donnera l'hypothèse (oie ou cygne) la plus probable, c'est-à-dire la meilleure pour résumer les connaissances.

La règle de Bayes, connue de tous les étudiants en mathématiques, permet d'affirmer que :

$$P(\text{oie} \mid \text{sombre}) = \frac{P(\text{sombre} \mid \text{oie})P(\text{oie})}{P(\text{sombre})}$$

$$P(\text{cygne} \mid \text{sombre}) = \frac{P(\text{sombre} \mid \text{cygne})P(\text{cygne})}{P(\text{sombre})}$$

Donc :

$$\frac{P(\text{cygne} \mid \text{sombre})}{P(\text{oie} \mid \text{sombre})} = \frac{P(\text{sombre} \mid \text{cygne})P(\text{cygne})}{P(\text{sombre} \mid \text{oie})P(\text{oie})}$$

et avec les valeurs numériques de notre exemple :

$$\frac{P(\text{cygne} \mid \text{sombre})}{P(\text{oie} \mid \text{sombre})} = \frac{(1/20).(3/4)}{(9/10).(1/4)} = \frac{1}{6}$$

Il y a six chances sur sept que l'oiseau soit une oie : un pari à prendre à six contre un.

Dans cette histoire, il a été question de décision, mais pas d'apprentissage. Où est-il caché ? En réalité, il est complètement résumé par les chiffres que donne l'expert. Par conséquent, ce chapitre développera non seulement les aspects de décision grâce à la règle de Bayes, mais surtout la manière dont l'expert a réussi à apprendre ces chiffres à partir d'exemples, de façon à ce que cette règle de décision soit la plus efficace possible.

14.1 L'apprentissage bayésien

14.1.1 Présentation

L'introduction générale à l'apprentissage bayésien a été faite au chapitre 2. Rappelons que son but est de choisir l'hypothèse $h \in \mathcal{H}$ qui minimise en moyenne l'erreur commise vis-à-vis de la fonction cible f , après observation de l'ensemble d'apprentissage \mathcal{S} .

On possède un échantillon d'exemples supervisés :

$$\mathcal{S} = \{(\mathbf{x}_1, \mathbf{u}_1), \dots, (\mathbf{x}_i, \mathbf{u}_i), \dots, (\mathbf{x}_m, \mathbf{u}_m)\}$$

On cherche à apprendre la meilleure hypothèse $h \in \mathcal{H}$ pour les expliquer, c'est-à-dire à trouver l'application de \mathcal{X} dans \mathcal{U} qui généralise en moyenne le mieux les couples de \mathcal{S} . On suppose ici que l'ensemble \mathcal{X} des valeurs \mathbf{x}_i est numérique. En revanche, les \mathbf{u}_i sont des valeurs de supervision qui peuvent être de natures variées. Par exemple :

- Si $\mathcal{X} = \mathbb{R}^d$ et $\mathbf{u}_i \in \{\omega_1, \dots, \omega_C\}$, on veut apprendre une règle de classification.
- Si $\mathcal{X} = \mathbb{R}^d$ et $\mathbf{u}_i \in [0, 1]$, on veut apprendre une fonction aléatoire.
- Si $\mathcal{X} = \mathbb{R}^d$ et $\mathbf{u}_i \in \mathbb{R}$, on veut apprendre une fonction réelle de d variables réelles.

La règle bayésienne d'apprentissage, ou décision bayésienne, consiste à élire comme meilleure hypothèse dans \mathcal{H} celle qui a la plus grande probabilité connaissant les données d'apprentissage. Il s'agit donc d'une méthode générale dont nous allons d'abord décrire le principe avant de l'instancier sur quelques exemples. L'apprentissage d'une règle de classification sera particulièrement détaillé, compte tenu de son importance pratique.

Pour des raisons de clarté, nous allons d'abord nous placer dans le cas où \mathcal{H} est un ensemble fini, par exemple un ensemble de deux classes comme $\{\text{oie}, \text{cygne}\}$.

Le cas où \mathcal{H} est fini

Commençons par ajouter provisoirement une nouvelle contrainte : supposons que l'on puisse définir une probabilité $P(\mathcal{S})$ de voir apparaître l'échantillon d'apprentissage \mathcal{S} . Cela est facile à faire si l'ensemble des échantillons possibles est lui aussi fini, mais c'est en général une hypothèse irréaliste. Mais peu importe, cette valeur disparaîtra rapidement des calculs.

Il est alors possible, puisque \mathcal{H} est aussi fini, de définir la probabilité $P(h)$ et la probabilité $P(h \mid \mathcal{S})$ de h sachant \mathcal{S} , ceci pour tout élément de \mathcal{H} .

La politique bayésienne consiste par définition à chercher l'hypothèse h^* la plus probable connaissant les exemples. h^* est définie par :

$$h^* = \operatorname{ArgMax}_{h \in \mathcal{H}} P(h | \mathcal{S})$$

La règle de Bayes nous dit que :

$$P(h, \mathcal{S}) = P(h | \mathcal{S})P(\mathcal{S}) = P(\mathcal{S} | h)P(h)$$

Définition 14.1

L'hypothèse h^ choisie par l'apprentissage bayésien est telle que :*

$$h^* = \operatorname{ArgMax}_{h \in \mathcal{H}} \frac{P(\mathcal{S} | h)P(h)}{P(\mathcal{S})} = \operatorname{ArgMax}_{h \in \mathcal{H}} P(\mathcal{S} | h)P(h)$$

La valeur $P(h)$ est appelée probabilité a priori de l'hypothèse h .

Une fois les données d'apprentissage observées et la règle de Bayes appliquée, la probabilité de l'hypothèse h devient $P(h | \mathcal{S})$: c'est la probabilité a posteriori de h .

La règle bayésienne d'apprentissage désigne donc dans \mathcal{H} la règle de plus forte probabilité a posteriori. Cette règle s'appelle aussi la règle MAP (maximum a posteriori).

Quand on suppose que les exemples sont des tirages indépendants (i.i.d.), ce qui est une hypothèse naturelle (voir le chapitre 2), on peut écrire :

$$P(\mathcal{S} | h) = P(h) \prod_{i=1}^m P(\mathbf{z}_i | h)$$

Dans ce cas, la règle bayésienne d'apprentissage MAP consiste à choisir h^* comme :

$$h^* = \operatorname{ArgMax}_{h \in \mathcal{H}} [P(h) \prod_{i=1}^m P(\mathbf{z}_i | h)]$$

ce qui peut aussi s'écrire :

$$h^* = \operatorname{ArgMin}_{h \in \mathcal{H}} [\operatorname{Log}(P(h)) + \sum_{i=1}^m \operatorname{Log}(P(\mathbf{z}_i | h))]$$

Il est courant de se placer dans un cas simplifié, si c'est justifié :

Définition 14.2

Si on suppose que toutes les hypothèses sont équiprobables dans \mathcal{H} , la règle MAP se simplifie et prend le nom de règle du maximum de vraisemblance :

$$h^* = \operatorname{ArgMax}_{h \in \mathcal{H}} \prod_{i=1}^m P(\mathbf{z}_i | h) = \operatorname{ArgMin}_{h \in \mathcal{H}} \sum_{i=1}^m \operatorname{Log}(P(\mathbf{z}_i | h))$$

Relâchons maintenant l'hypothèse peu crédible selon laquelle il est possible de définir une probabilité $P(\mathcal{S})$, mais gardons un cadre probabiliste en supposant qu'il existe une *densité* de probabilité $p(\mathcal{S})$ associée à chaque échantillon \mathcal{S} . On peut dans ce cas définir aussi $p(\mathcal{S} | h)$, la densité conditionnelle de \mathcal{S} connaissant l'hypothèse h .

Dans ce cas, la règle *MAP* s'écrit :

$$h^* = \operatorname{ArgMax}_{h \in \mathcal{H}} p(\mathcal{S} | h)P(h)$$

et la règle du maximum de vraisemblance :

$$h^* = \operatorname{ArgMin}_{h \in \mathcal{H}} \sum_{i=1}^m \operatorname{Log}(p(z_i | h))$$

Le cas où \mathcal{H} est infini

Dans ce cas, il n'est plus possible de définir une probabilité $P(h)$. Néanmoins, on peut supposer qu'il existe une distribution de probabilités sur \mathcal{H} et donc qu'une densité de probabilité $p(h)$ peut être définie pour tout h . De même, il existe des densités conditionnelles $p(h | \mathcal{S})$ et $p(\mathcal{S} | h)$.

On cherche alors l'hypothèse h^* telle que :

$$h^* = \operatorname{ArgMax}_{h \in \mathcal{H}} p(\mathcal{S} | h)p(h)$$

soit, avec la règle *MAP* :

$$h^* = \operatorname{ArgMin}_{h \in \mathcal{H}} [\operatorname{Log}(p(h)) + \sum_{i=1}^m \operatorname{Log}(p(z_i | h))]$$

14.1.2 Un petit retour en arrière

La règle bayésienne optimale a été présentée au chapitre 2 sous une forme légèrement différente, pour des raisons de cohérence des notations dans ce chapitre :

$$h^* = \operatorname{ArgMin}_{h \in \mathcal{H}} l(h|f) p_{\mathcal{F}}(f) p_{\mathcal{X}}(\mathbf{x}|f)$$

avec :

- $p_{\mathcal{F}}(f)$ est la densité de probabilité *a priori* de la fonction cible, c'est-à-dire que l'état réel du monde soit f .
- $p_{\mathcal{X}}(\mathbf{x}|f)$ est la probabilité de l'observation \mathbf{x} connaissant la fonction cible f .
- $l(h|f)$ est le coût de choisir h au lieu de la vraie fonction f .

Ici, nous utilisons la fonction de perte la plus simple (voir annexe 18.1) :

$$l(u_i, h(x_i)) = \begin{cases} 0 & \text{si } u_i = h(x_i) \\ 1 & \text{si } u_i \neq h(x_i) \end{cases} \quad (14.1)$$

14.1.3 L'apprentissage bayésien d'une règle de classification

La première application de l'apprentissage bayésien est la *reconnaissance statistique des formes* ou *l'apprentissage d'une règle de classification*: l'espace \mathcal{X} est égal à \mathbb{R}^d et on cherche à apprendre une application de \mathbb{R}^d dans $\{\omega_1, \dots, \omega_C\}$. On suppose qu'à chaque classe ω_i il est possible d'associer une probabilité *a priori* $P(\omega_i)$ et que tout vecteur correspondant à un point de la classe ω_i peut être considéré comme le résultat d'un tirage aléatoire indépendant de densité $p(\mathbf{x} | \omega_i)$.

Si on veut utiliser les données d'apprentissage pour attribuer avec la meilleure vraisemblance une classe à un point \mathbf{x} quelconque, il faut calculer la probabilité *a posteriori* $P(\omega_i \mid \mathbf{x})$ d'observer ce point conditionnellement à chaque classe; ce calcul se fait, à partir des valeurs précédentes, par la règle de Bayes¹:

$$P(\omega_i \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \omega_i)P(\omega_i)}{p(\mathbf{x})} \quad (14.2)$$

La décision par la règle du maximum *a posteriori* (*MAP*) s'écrit alors en cherchant la classe ω^* de plus grande probabilité *a posteriori*:

$$\omega^* = \operatorname{ArgMax}_{i=1,C} p(\mathbf{x} \mid \omega_i)P(\omega_i) \quad (14.3)$$

Le problème de l'apprentissage d'une règle de classification serait donc résolu si l'on connaissait $P(\omega_i)$ et $p(\mathbf{x} \mid \omega_i)$. Comme on le verra au paragraphe 14.1.5, ce problème se ramène essentiellement à estimer chaque $p(\mathbf{x} \mid \omega_i)$ à partir des échantillons d'apprentissage supervisés par la classe ω_i .

La séparation des classes peut aussi se formuler de manière géométrique, comme présenté au chapitre 3.

Définition 14.3

*On appelle surface séparatrice Υ_{ij} entre les deux classes ω_i et ω_j le lieu des points où la probabilité *a posteriori* d'appartenir à ω_i et à ω_j est égale.*

$$\Upsilon_{ij} \text{ est définie par: } P(\omega_i \mid \mathbf{x}) = P(\omega_j \mid \mathbf{x})$$

14.1.4 La classification bayésienne est optimale en moyenne...

On a donc défini la *règle de classification bayésienne* h^* en attribuant au point \mathbf{x} la classe ω^* qui a la plus forte probabilité conditionnellement à \mathbf{x} parmi toutes les classes :

$$h^* \text{ attribue la classe } \omega^* = \operatorname{ArgMax}_{i \in \{1, \dots, C\}} [P(\omega_i \mid \mathbf{x})] = \operatorname{ArgMax}_{i \in \{1, \dots, C\}} [p(\mathbf{x} \mid \omega_i)P(\omega_i)] \quad (14.4)$$

Cette règle est *optimale en moyenne*: parmi toutes les règles de classification possibles, elle est celle qui minimise la probabilité d'erreur, connaissant la probabilité *a priori* des classes ω_i .

Ce résultat nous est déjà connu: on a déjà traité aux chapitres 2 et 3 de cette notion de probabilité d'erreur et on a vu que la règle de classification bayésienne minimise l'espérance (la moyenne statistique) de mauvaise classification de l'objet \mathbf{x} quand il parcourt \mathcal{X} .

La valeur $R(h^*)$ est appelée *erreur bayésienne de classification*; les algorithmes d'apprentissage de règles de classification visent donc souvent à l'approcher. Une démonstration directe de cette optimalité est donnée en annexe 18.8.

14.1.5 ...mais on ne peut que l'approcher.

Le problème de l'apprentissage d'une règle de classification serait résolu, sous les hypothèses précédentes, si l'on possédait une connaissance exacte pour chaque classe de la probabilité *a*

1. Cette formule mélange probabilités et densités de probabilités; il est facile de vérifier sa validité en partant de sa forme originale $P(\omega_i \mid \mathbf{x}) = \frac{P(\mathbf{x} \in V \mid \omega_i)P(\omega_i)}{P(\mathbf{x} \in V)}$. V est un volume fini de \mathbb{R}^d et on a l'égalité: $P(\mathbf{x} \in V) = \int_V p(\mathbf{x})d\mathbf{x}$.

priori $P(\omega_i)$ et de la densité conditionnelle $p(\mathbf{x} \mid \omega_i)$. Mais on ne dispose en pratique que des données d'apprentissage $\mathcal{S} = \{(\mathbf{x}_1, \mathbf{u}_1), \dots, (\mathbf{x}_m, \mathbf{u}_m)\}$.

Considérons l'une après l'autre les deux valeurs qui interviennent dans la formule

$$h^* \text{ attribue la classe } \omega^* = \underset{i \in \{1, \dots, C\}}{\text{ArgMax}} [p(\mathbf{x} \mid \omega_i)P(\omega_i)]$$

L'estimation de la probabilité *a priori* des classes

Pour estimer les valeurs $P(\omega_i)$, les probabilités *a priori* des classes, on peut :

- soit, en l'absence d'information particulière, les supposer égales et donc prendre l'estimateur : $\widehat{P(\omega_i)} = \frac{1}{C}$.
- soit supposer l'échantillon d'apprentissage représentatif et les estimer par les fréquences d'apparition de chaque classe dans cet ensemble² : $\widehat{P(\omega_i)} = \frac{m_i}{m}$.
- soit utiliser un estimateur intermédiaire (formule de Laplace)

$$\widehat{P(\omega_i)} = \frac{m_i + M/C}{m + M}$$

où M est un nombre arbitraire. Cette formule est employée quand m est petit, donc quand les estimations m_i/m sont très imprécises. M représente une augmentation virtuelle du nombre d'exemples, pour lesquels on suppose les classes équiprobables.

Le premier cas s'applique par exemple à la reconnaissance des chiffres manuscrits sur les chèques ; en revanche, pour les codes postaux, la seconde méthode est préférable (la proportion de chiffres 0 y est supérieure à celle des autres).

Le second cas semble plus naturel mais il peut aussi être trompeur : dans certains problèmes, les classes ne sont pas représentées de la même manière dans l'ensemble d'apprentissage et dans les exemples qu'il faudra classer. Par exemple, un diagnostic médical peut s'apprendre à partir d'un ensemble d'apprentissage comportant un nombre équilibré d'exemples et de contre-exemples, alors que la maladie est rare.

L'estimation des densités conditionnelles *a priori*

Il reste donc à estimer les densités $p(\mathbf{x} \mid \omega_i)$. Dans un problème d'apprentissage de règle de classification, on dispose d'un échantillon d'exemples supervisés : le problème se ramène donc à estimer chaque $p(\mathbf{x} \mid \omega_i)$ à partir des échantillons d'apprentissage supervisés par la classe ω_i .

Indépendamment pour chaque classe, on se trouve donc finalement à devoir estimer une densité de probabilité à partir d'un nombre fini d'observations. C'est un problème tout à fait classique en statistiques. Il faut introduire des hypothèses supplémentaires pour le résoudre (un biais, en terme d'apprentissage artificiel). On a l'habitude de distinguer (voir le chapitre 2) :

- Les méthodes *paramétriques*, où l'on suppose que les $p(\mathbf{x} \mid \omega_i)$ possèdent une certaine forme analytique ; en général, on fait l'hypothèse qu'elles sont des distributions gaussiennes. Dans ce cas, le problème se ramène à estimer la moyenne et la covariance de chaque distribution ; la probabilité d'appartenance d'un point \mathbf{x} à une classe se calcule alors directement à partir des coordonnées de \mathbf{x} . Ce sera l'objet du paragraphe 14.2.2.
- Les méthodes *non paramétriques*, pour lesquelles on estime localement les densités $p(\mathbf{x} \mid \omega_i)$ au point \mathbf{x} en observant l'ensemble d'apprentissage autour de ce point. Ces méthodes sont implémentées par la technique des *fenêtres de Parzen* (paragraphe 14.3.2) ou l'algorithme des *k-plus proches voisins* (paragraphe 14.3.3).

2. Rappelons que le nombre d'exemples total est noté m , alors que m_i désigne le nombre d'exemples supervisés par la classe i .

- Les méthodes semi paramétriques, pour lesquelles nous ne connaissons pas non plus la forme analytique des distributions de probabilités. Nous supposons cependant que ces distributions appartiennent à des familles et que les « hyper paramètres » qui les caractérisent à l'intérieur de cette famille peuvent être déterminés.

14.1.6 La règle bayésienne et la régression aux moindres carrés

Considérons maintenant le problème de la régression, c'est-à-dire de l'apprentissage d'une fonction $f : \mathcal{X} \rightarrow \mathbb{R}$ à partir d'un ensemble d'exemples \mathcal{S} :

$$\mathcal{S} = \{(\mathbf{x}_1, \mathbf{u}_1), \dots, (\mathbf{x}_i, \mathbf{u}_i), \dots, (\mathbf{x}_m, \mathbf{u}_m)\}$$

Nous sommes ici dans le cas où l'espace des hypothèses et celui des échantillons possibles sont tous deux infinis: nous emploierons donc des densités de probabilité.

Plaçons-nous d'autre part sous l'hypothèse que la partie supervision de chaque exemple s'écrit :

$$\mathbf{u}_i = f(\mathbf{x}_i) + \mathbf{e}_i$$

où \mathbf{e}_i est un tirage aléatoire d'une distribution gaussienne de moyenne nulle et de variance inconnue σ^2 . En d'autres termes, on considère qu'il s'agit d'apprendre la fonction f à partir d'exemples bruités³.

On cherche maintenant dans un ensemble de fonctions hypothèses \mathcal{H} quelle est h^* , la plus probable d'être la fonction f . On sait que :

$$h^* = \operatorname{ArgMax}_{h \in \mathcal{H}} p(h \mid \mathcal{S})p(h)$$

Si on suppose toutes les fonctions équiprobables dans \mathcal{H} et les exemples indépendants, on peut appliquer la règle du maximum de vraisemblance :

$$h^* = \operatorname{ArgMax}_{h \in \mathcal{H}} p(\mathcal{S} \mid h) = \prod \operatorname{ArgMax}_{h \in \mathcal{H}} p(\mathbf{z}_i \mid h)$$

La relation :

$$\mathbf{u}_i = f(\mathbf{x}_i) + \mathbf{e}_i$$

contraint la valeur \mathbf{u}_i à être le résultat d'un tirage aléatoire d'une distribution gaussienne de centre $f(\mathbf{x}_i)$ et de variance σ^2 . Par conséquent, $p(\mathbf{u}_i \mid h)$ est le résultat d'un tirage aléatoire d'une distribution gaussienne de centre $h(\mathbf{x}_i)$ et de variance σ^2 . Ces contraintes sur la relation entre les \mathbf{x}_i et les \mathbf{u}_i étant ainsi formalisées, l'apprentissage par maximum de vraisemblance se transforme en :

$$h^* = \operatorname{ArgMax}_{h \in \mathcal{H}} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(\mathbf{u}_i - h(\mathbf{x}_i))^2}$$

d'où, en prenant le logarithme :

$$h^* = \operatorname{ArgMin}_{h \in \mathcal{H}} \sum_{i=1}^m \frac{1}{2\sigma^2} (\mathbf{u}_i - h(\mathbf{x}_i))^2 = \operatorname{ArgMin}_{h \in \mathcal{H}} \sum_{i=1}^m (\mathbf{u}_i - h(\mathbf{x}_i))^2 \quad (14.5)$$

Par conséquent, l'hypothèse h^* la plus probable dans \mathcal{H} est celle qui minimise la somme sur tous les exemples \mathbf{x}_i du carré de la différence entre la valeur $h^*(\mathbf{x}_i)$ au point \mathbf{x}_i et la valeur bruitée de la fonction $f(\mathbf{x}_i)$, donnée par $\mathbf{u}_i = f(\mathbf{x}_i) + \mathbf{e}_i$.

3. Cette hypothèse est souvent réaliste et en particulier très employée dans le traitement du signal et des images.

h^* est donc la meilleure fonction à la fois au sens de la minimisation des moindres carrés et au sens de la règle du maximum de vraisemblance, sous l'hypothèse que le bruit de mesure sur les exemples est gaussien.

Un exemple classique est celui de la régression linéaire dans un plan : la meilleure droite pour approximer un ensemble de points du plan (si l'on suppose ces points issus d'une même droite, mais déplacés par un bruit gaussien) se calcule comme celle qui minimise la somme des distances des points à elle-même (figure 14.1).

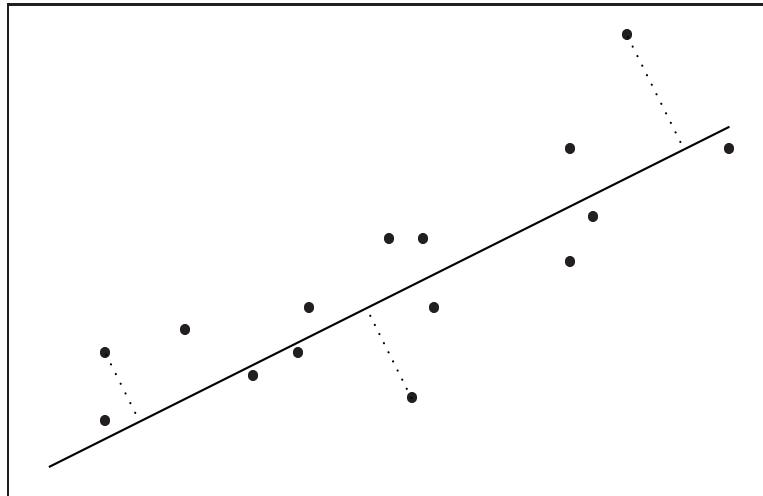


FIG. 14.1 – La meilleure droite pour approximer un ensemble de points.

14.1.7 La règle bayésienne et la minimisation de l'entropie croisée

Plaçons nous maintenant dans un autre cas d'apprentissage : celui d'une fonction aléatoire à deux valeurs $f : \mathbb{R} \rightarrow \{0, 1\}$.

Les données d'apprentissage qui résument la connaissance sur f sont rassemblées dans l'ensemble $\mathcal{S} = \{(\mathbf{x}_1, \mathbf{u}_1), \dots, (\mathbf{x}_i, \mathbf{u}_i), \dots, (\mathbf{x}_m, \mathbf{u}_m)\}$, avec $\mathbf{u}_i = f(\mathbf{x}_i) = 0$ ou 1 .

L'espace des hypothèses est un ensemble \mathcal{H} dans lequel on cherche la meilleure fonction h^* , au sens bayésien, pour approcher f à partir de \mathcal{S} .

La probabilité que les données aient été engendrées par la distribution de probabilités h s'écrit, en supposant les données indépendantes entre elles :

$$P(\mathcal{S} | h) = \prod_{i=1}^m P(\mathbf{z}_i | h)$$

En utilisant la règle des probabilités conditionnelles :

$$P(\mathbf{z}_i | h) = P(\mathbf{x}_i, \mathbf{u}_i | h) = P(\mathbf{u}_i | h, \mathbf{x}_i)P(\mathbf{x}_i)$$

$$P(\mathcal{S} | h) = \prod_{i=1}^m P(\mathbf{u}_i | h, \mathbf{x}_i)P(\mathbf{x}_i)$$

En notant que $P(\mathbf{u}_i | h, \mathbf{x}_i)$ vaut $h(\mathbf{x}_i)$ si $\mathbf{u}_i = 1$ et $1 - h(\mathbf{x}_i)$ si $\mathbf{u}_i = 0$, on peut écrire :

$$P(\mathbf{u}_i | h, \mathbf{x}_i) = h(\mathbf{x}_i)^{\mathbf{u}_i}(1 - h(\mathbf{x}_i))^{1-\mathbf{u}_i}$$

D'où :

$$P(\mathcal{S} \mid h) = \prod_{i=1}^m h(\mathbf{x}_i)^{\mathbf{u}_i} (1 - h(\mathbf{x}_i))^{1-\mathbf{u}_i} P(\mathbf{x}_i)$$

La règle bayésienne d'apprentissage h^* s'écrit donc :

$$h^* = \underset{h \in \mathcal{H}}{\text{ArgMax}} \quad P(\mathcal{S} \mid h) = \underset{h \in \mathcal{H}}{\text{ArgMax}} \quad \prod_{i=1}^m h(\mathbf{x}_i)^{\mathbf{u}_i} (1 - h(\mathbf{x}_i))^{1-\mathbf{u}_i} P(\mathbf{x}_i) \quad (14.6)$$

$P(\mathbf{x}_i)$ peut être supposé indépendant de h : il est raisonnable de supposer que la probabilité d'observer telle ou telle donnée est indépendante de l'hypothèse que l'on fait. Par conséquent :

$$h^* = \underset{h \in \mathcal{H}}{\text{ArgMax}} \quad \prod_{i=1}^m h(\mathbf{x}_i)^{\mathbf{u}_i} (1 - h(\mathbf{x}_i))^{1-\mathbf{u}_i} \quad (14.7)$$

D'où, en passant au logarithme :

$$h^* = \underset{h \in \mathcal{H}}{\text{ArgMax}} \quad \sum_{i=1}^m \mathbf{u}_i \text{ Log}[h(\mathbf{x}_i)] + (1 - \mathbf{u}_i)(1 - h(\mathbf{x}_i)) \quad (14.8)$$

Le terme de droite est l'opposé de l'*entropie croisée* entre la distribution des exemples sur $\{0, 1\}$ et la distribution h . Par conséquent, la meilleure fonction aléatoire au sens bayésien est celle qui maximise ce terme. Rappelons que l'entropie croisée a été utilisée pour l'apprentissage des arbres de décision : chaque choix pendant la phase de construction est donc localement optimal au sens bayésien (voir le chapitre 11).

14.1.8 La règle bayésienne et la longueur minimale de description

Nous anticipons sur un aspect qui sera abordé au chapitre 17: les rapports entre le principe *MDL* et l'apprentissage bayésien. On peut en effet interpréter la décision bayésienne comme un compromis optimal entre la qualité et la complexité de la solution. Pour cela, il faut raisonner en termes de codage. Rappelons le premier théorème de Shannon ([Cov91]), qui assure le résultat suivant :

Théorème 14.1

Soit un ensemble fini d'objets $\{O_1, \dots, O_n\}$, chacun pouvant apparaître avec la probabilité P_1, \dots, P_n . Un code binaire assigne à chaque objet un message sous la forme d'une séquence de bits, de manière bi-univoque. Le code binaire optimal, c'est-à-dire celui qui minimise en moyenne la longueur du message transmis, nécessite $-\log_2 P_i$ bits pour transmettre le message signifiant que l'objet O_i est apparu.

Nous supposons ici \mathcal{H} de taille finie et nous notons Γ le code optimal correspondant à la distribution des hypothèses dans \mathcal{H} . La transmission de l'hypothèse h_i par le code Γ nécessite un nombre de bits que nous notons L_Γ et qui vaut :

$$L_\Gamma(h_i) = -\log_2 P(h_i)$$

De même, si nous voulons transmettre les données \mathcal{S} en supposant que l'émetteur et le récepteur connaissent tous les deux l'hypothèse h_i , il faut choisir un code optimal correspondant

à la distribution conditionnelle des événements $P(\mathcal{S} \mid h_i)$. Notons ce code Λ . La longueur du message est alors :

$$L_\Lambda(\mathcal{S} \mid h_i) = -\log_2 P(\mathcal{S} \mid h_i)$$

Maintenant, pour transmettre à la fois les données et une hypothèse h_i , il nous faut donc au mieux :

$$L(h_i, \mathcal{S}) = L_\Gamma(h_i) + L_\Lambda(\mathcal{S} \mid h_i) = -\log_2 P(h_i) - \log_2 P(\mathcal{S} \mid h_i)$$

On appelle souvent $L(h_i, \mathcal{S})$ la *longueur de description* de la solution h_i . Elle reflète d'une certaine façon la qualité de h_i , puisqu'elle s'interprète comme le coût de transmettre à la fois une hypothèse et les données connaissant cette hypothèse. Pour expliquer ceci, prenons un exemple.

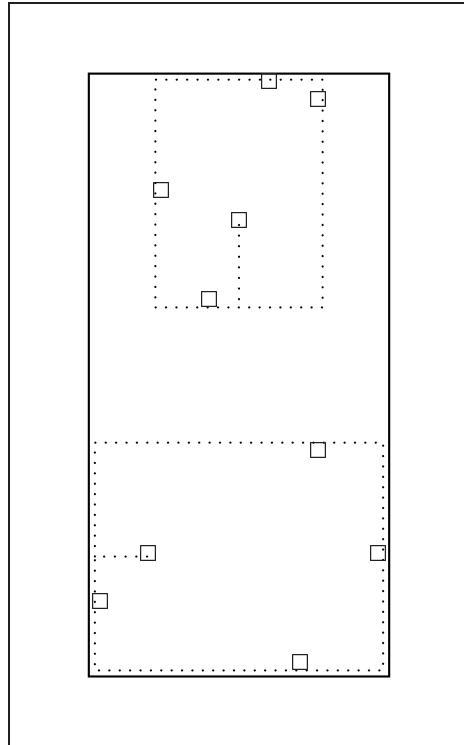


FIG. 14.2 – L'hypothèse constituée d'un seul rectangle (en trait continu) est courte à coder, mais sa précision n'est pas bonne : beaucoup de points sont loin de la frontière. Celle constituée de deux rectangles (en pointillés) est à la fois assez courte à coder et précise : seuls deux points ne sont pas sur la frontière. Une troisième hypothèse est celle constituée de rectangles entourant chaque point : elle est très précise, mais son codage est le plus long. Un bon compromis est donc l'hypothèse constituée de deux rectangles.

Un exemple

Imaginons de disposer de données \mathcal{S} qui sont des points de \mathbb{R}^2 et supposons qu'une hypothèse h se décrive comme une union de rectangles contenant les données.

Il existe un code optimal Γ pour calculer la longueur de description $L_\Gamma(h)$ d'une hypothèse h , que nous ne connaissons pas. Il est cependant vraisemblable que coder chaque rectangle par les coordonnées de ses deux sommets opposés en diagonale soit une bonne approche. Le nombre de rectangles est donc sans doute relié directement à la complexité du codage de l'hypothèse.

Nous pouvons mesurer la qualité de la description d'une donnée en observant si elle est proche ou non de la frontière d'un rectangle qui la contient. Pour cela, faisons passer une droite horizontale et une droite verticale par ce point et mesurons à quelle distance à gauche, à droite, en haut et en bas elle rencontre la première frontière du rectangle. La plus faible de ces quatre valeurs nous donne une idée de la qualité de la description de ce point : il est satisfaisant qu'une frontière de rectangle passe près du point. Nous pouvons sommer une telle valeur sur tous les points et en prendre l'inverse : nous disposons d'une mesure de qualité d'une hypothèse connaissant les données. Le codage de cette hypothèse, à supposer qu'on connaisse le code optimal Λ , prend exactement la longueur de description $L_\Lambda(\mathcal{S} | h)$.

Regardons maintenant deux cas extrêmes :

- On choisit comme hypothèse un rectangle unique, couvrant juste les données. Son codage $L_\Gamma(h_i)$ sera très court, mais la valeur de $Q(\mathcal{S}, h)$ est en revanche élevée. Au total, $L(h_i, \mathcal{S}) = L_\Gamma(h_i) + L_\Lambda(\mathcal{S} | h_i)$ sera forte.
- On choisit comme hypothèse l'union de tout petits rectangles, chacun entourant une donnée. Cette fois la valeur de $Q(\mathcal{S}, h)$ sera excellente, mais le codage de l'hypothèse sera long. Au final, $L(h_i, \mathcal{S})$ sera forte pour les raisons inverses.

En réalité, il existe un compromis pour lequel $L(h_i, \mathcal{S})$ est minimale. La figure 14.2 en donne une idée sur un exemple.

En revenant au cas général, l'hypothèse ayant la longueur de description minimale est donc en un certain sens la meilleure hypothèse. Elle est caractérisée par :

$$h^* = \underset{h \in \mathcal{H}}{\text{ArgMin}} [-\log_2 P(h) - \log_2 P(\mathcal{S} | h)]$$

En changeant le signe et en prenant l'exponentielle, on constate que h^* est également la meilleure hypothèse bayésienne :

$$h^* = \underset{h \in \mathcal{H}}{\text{ArgMax}} P(\mathcal{S} | h)P(h) \quad (14.9)$$

14.1.9 L'apprentissage bayésien non supervisé

Un autre aspect de l'apprentissage bayésien est sa capacité à être utilisé en apprentissage non supervisé, c'est-à-dire à segmenter des données en familles homogènes. Nous traitons ce cas au chapitre 15, intitulé : « La classification non supervisée et la découverte automatique ».

14.2 Les méthodes paramétriques

L'une des approches classiques pour estimer des distributions de probabilités est de les représenter à l'aide de fonctions paramétrées. Il s'agit alors d'optimiser la valeur de ces paramètres pour que les fonctions s'adaptent aux données d'apprentissage. On peut alors appliquer la règle de décision bayésienne en remplaçant les probabilités vraies par leur estimation. Le chapitre 2 a déjà abordé ce sujet.

En classification, il s'agit donc d'estimer les grandeurs $P(\omega_i)$ et $p(\mathbf{x}|\omega_i)$ pour chaque classe ω_i . Nous avons déjà appris à estimer $P(\omega_i)$, où m_i est le nombre de formes appartenant à la classe ω_i et m le nombre total de formes observées (paragraphe 14.1.5). Reste à estimer $p(\mathbf{x}|\omega_i)$.

14.2.1 L'estimation par maximum de vraisemblance

Nous allons supposer que la fonction de densité de probabilité $p(\mathbf{x}|\omega)$ dépend d'un ensemble de paramètres que nous notons $\boldsymbol{\theta} = (\theta_1, \dots, \theta_L)^\top$. L'ensemble des valeurs que peut prendre $\boldsymbol{\theta}$

est noté Θ . Dans le cas d'un problème de classification, nous aurons une fonction par classe afin de représenter $p(\mathbf{x}|\omega_i)$ pour chaque classe ω_i , ou plus précisément, dans cette approche paramétrique, $p(\mathbf{x}|\boldsymbol{\theta}_i)$.

Nous disposons d'un échantillon de données $\mathcal{S} = \{\mathbf{x}_1, \dots, \mathbf{x}_m\}$ de taille m dont nous extrayons le sous-ensemble de \mathcal{S}^i de taille m_i relatif à une classe ω_i :

$$\mathcal{S}^i = \{\mathbf{x}_1, \dots, \mathbf{x}_{m_i}\}$$

En supposant que ces données soient tirées indépendamment les unes des autres suivant la loi de distribution $p(\mathbf{x}|\boldsymbol{\theta}_i)$, la densité de probabilité de l'échantillon total s'écrit :

$$\mathcal{L}(\boldsymbol{\theta}_i) = p(\mathcal{S}^i|\boldsymbol{\theta}_i) = \prod_{i=1}^{m_i} p(\mathbf{x}_i|\boldsymbol{\theta}_i)$$

où $\mathcal{L}(\boldsymbol{\theta}_i)$ est la *vraisemblance* du vecteur de paramètres $\boldsymbol{\theta}_i$ pour l'échantillon \mathcal{S}^i .

La méthode du maximum de vraisemblance consiste à prendre pour estimation $\hat{\boldsymbol{\theta}}_i$ la valeur du vecteur de paramètres inconnu $\boldsymbol{\theta}_i$ qui maximise la vraisemblance que les données aient été produites à partir de la distribution $p(\mathbf{x}|\boldsymbol{\theta}_i)$.

$$\hat{\boldsymbol{\theta}}_i = \underset{\boldsymbol{\theta}_i \in \Theta}{\text{ArgMax}} \mathcal{L}(\boldsymbol{\theta}_i)$$

Il est plus facile de minimiser l'opposé du logarithme de cette expression, d'où :

$$\begin{aligned} \hat{\boldsymbol{\theta}}_i &= \underset{\boldsymbol{\theta}_i \in \Theta}{\text{ArgMin}} \{-\log \mathcal{L}(\boldsymbol{\theta}_i)\} \\ &= \underset{\boldsymbol{\theta}_i \in \Theta}{\text{ArgMin}} \left\{ -\sum_{i=1}^{m_i} \ln p(\mathbf{x}_i|\boldsymbol{\theta}_i) \right\} \end{aligned} \quad (14.10)$$

Il est à noter que les estimateurs obtenus par la méthode du maximum de vraisemblance sont excellents⁴. Cependant cela n'en garantit pas la qualité pour des échantillons de taille réduite.

Pour la plupart des choix de fonctions de densité, l'optimum $\hat{\boldsymbol{\theta}}_i$ devra être estimé en utilisant des procédures numériques itératives (voir par exemple le chapitre 3). Pour certaines formes fonctionnelles, notamment pour le cas particulier des densités normales (gaussiennes), la solution optimale peut être déterminée de manière analytique, en cherchant la valeur de $\boldsymbol{\theta}$ qui annule la dérivée de l'équation 14.10.

14.2.2 L'estimation des paramètres d'une distribution gaussienne

On suppose ici que chaque classe possède une distribution de probabilité de forme paramétrique. On traite en général uniquement le cas gaussien, c'est-à-dire que l'on suppose que la distribution de probabilité de chaque classe est une loi normale, entièrement définie par l'ensemble des paramètres $\boldsymbol{\theta} = (\mu, Q)^\top$ composé de son vecteur moyen et de sa matrice de covariance.

Faire une telle hypothèse est un biais fort dont la validité peut éventuellement être contrôlée par un test statistique; il faut garder à l'esprit que cette supposition est dépendante du choix de l'espace de représentation et ne possède aucune justification théorique *a priori*⁵. Mais elle

4. En termes techniques, asymptotiquement sans biais et de variance minimale.

5. La loi des grands nombres est souvent invoquée à tort pour justifier ce biais.

permet d'obtenir une solution analytique simple et un algorithme peu complexe d'apprentissage inductif dans \mathbb{R}^d .

Rappelons que la *moyenne* $\boldsymbol{\mu}$ d'une densité de probabilité p dans \mathbb{R}^d est un vecteur de dimension d et sa covariance une matrice $Q(d \times d)$. Si $E[p]$ dénote l'*espérance mathématique* de la variable aléatoire p , on a :

$$\begin{aligned}\boldsymbol{\mu} &= E[\mathbf{x}] \\ Q &= E[(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T]\end{aligned}$$

Une distribution de probabilité gaussienne a pour caractéristique de pouvoir entièrement être définie par son vecteur moyenne et sa matrice de covariance. En supposant donc la classe ω_i gaussienne, sa densité de probabilité s'écrit, dans un espace multidimensionnel :

$$p(\mathbf{x} | \omega_i) = \frac{|Q|^{-1/2}}{(2\pi)^{d/2}} \exp \left\{ -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_i)^T Q_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i) \right\} \quad (14.11)$$

Ce n'est qu'une généralisation de la définition en dimension $d = 1$, plus familière :

$$p(\mathbf{x} | \omega_i) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left\{ -\frac{(\mathbf{x} - \boldsymbol{\mu}_i)^2}{2\sigma^2} \right\} \quad (14.12)$$

Rappelons aussi (voir la figure 14.3) que la surface d'équidensité d'une distribution gaussienne est une quadrique (à deux dimensions, c'est une ellipse).

Compte tenu des m_i points d'apprentissage $\mathcal{S}^i = \{\mathbf{x}_1, \dots, \mathbf{x}_j, \dots, \mathbf{x}_{m_i}\}$, relatifs à la classe w_i (supposée gaussienne), il est démontré en annexe 18.5 que les meilleures estimations de sa moyenne $\boldsymbol{\mu}_i$ et de sa matrice de covariance Q_i au sens du maximum de vraisemblance (c'est-à-dire celles qui maximisent la probabilité d'observer les données d'apprentissage) se calculent simplement par :

$$\widehat{\boldsymbol{\mu}}_i = \frac{1}{m_i} \sum_{j=1}^{m_i} \mathbf{x}_j \quad (14.13)$$

$$\widehat{Q}_i = \frac{1}{m_i} \sum_{j=1}^{m_i} (\mathbf{x}_j - \widehat{\boldsymbol{\mu}}_i)(\mathbf{x}_j - \widehat{\boldsymbol{\mu}}_i)^T \quad (14.14)$$

14.2.2.1 Le résultat de l'apprentissage

On peut facilement interpréter dans le cas gaussien la règle bayésienne de décision en terme de surfaces séparatrices ; en effet, le lieu des points où les probabilités d'appartenir aux deux classes ω_i et ω_j sont égales a pour équation :

$$\begin{aligned}p(\mathbf{x} | \omega_i) &= \frac{|Q_i|^{-1/2}}{2\pi^{d/2}} \exp \left\{ -1/2 (\mathbf{x} - \boldsymbol{\mu}_i)^T Q_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i) \right\} \\ &= p(\mathbf{x} | \omega_j) = \frac{|Q_j|^{-1/2}}{2\pi^{d/2}} \exp \left\{ -1/2 (\mathbf{x} - \boldsymbol{\mu}_j)^T Q_j^{-1} (\mathbf{x} - \boldsymbol{\mu}_j) \right\}\end{aligned}$$

Après simplification et passage au logarithme, on obtient une forme quadratique du type :

$$\mathbf{x}^T \Phi \mathbf{x} + \mathbf{x}^T \phi + \alpha = 0 \quad (14.15)$$

où la matrice Φ , le vecteur ϕ et la constante α ne dépendent que de $\boldsymbol{\mu}_i$, $\boldsymbol{\mu}_j$, Q_i et Q_j . On constate donc que faire une hypothèse gaussienne sur la répartition de chaque classe revient à supposer des surfaces de décision quadriques ; à deux dimensions, ce sont des coniques. La figure 14.3 montre un exemple de cette propriété.

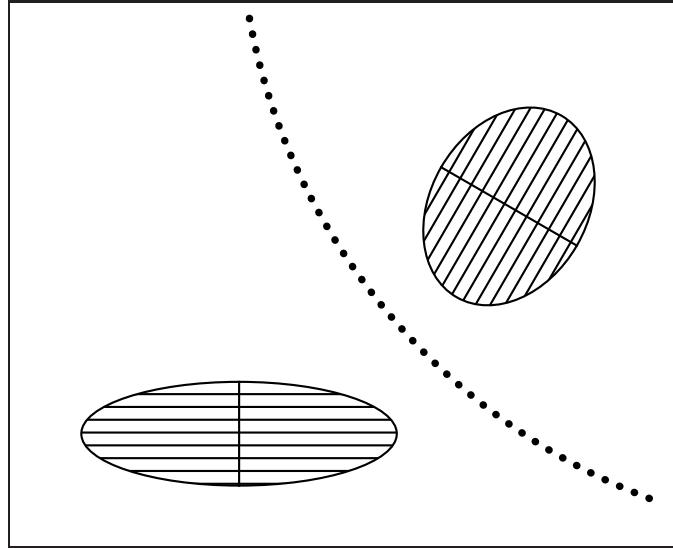


FIG. 14.3 – À deux dimensions, la surface séparatrice de deux classes gaussiennes bidimensionnelles est une conique (ici une branche d'hyperbole en pointillés) et les surfaces d'équidensité d'une distribution gaussienne sont des ellipses. On a représenté pour chaque classe l'ellipse d'équidensité telle que la probabilité d'appartenir à la classe soit supérieure à 0.5 quand on est à l'intérieur de cette ellipse.

14.2.2.2 Un exemple à deux dimensions

Considérons l'ensemble d'apprentissage suivant, comportant huit exemples, quatre pour chacune des deux classes :

$$\mathcal{S} = \left\{ \left(\begin{pmatrix} 0 \\ 4 \end{pmatrix}, \omega_1 \right), \left(\begin{pmatrix} 1 \\ 1 \end{pmatrix}, \omega_1 \right), \left(\begin{pmatrix} 3 \\ 3 \end{pmatrix}, \omega_1 \right), \left(\begin{pmatrix} 4 \\ 0 \end{pmatrix}, \omega_1 \right), \left(\begin{pmatrix} 4 \\ 0 \end{pmatrix}, \omega_2 \right), \left(\begin{pmatrix} 7 \\ 1 \end{pmatrix}, \omega_2 \right), \left(\begin{pmatrix} 8 \\ 4 \end{pmatrix}, \omega_2 \right), \left(\begin{pmatrix} 5 \\ 3 \end{pmatrix}, \omega_2 \right) \right\}$$

La modélisation gaussienne de la classe ω_1 amène les paramètres suivants, en notant \mathbf{x}_{1j} les vecteurs des exemples de cette classe :

$$\widehat{\boldsymbol{\mu}}_1 = \frac{1}{4} \sum_{j=1}^4 \mathbf{x}_{1j} = \frac{1}{4} \begin{pmatrix} 0 + 1 + 3 + 4 \\ 4 + 1 + 3 + 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$

$$\widehat{Q}_1 = \frac{1}{4} \sum_{j=1}^4 (\mathbf{x}_{1j} - \widehat{\boldsymbol{\mu}}_1)(\mathbf{x}_{1j} - \widehat{\boldsymbol{\mu}}_1)^T$$

Pour $j = 1$, par exemple, le terme de cette somme vaut :

$$\left[\begin{pmatrix} 0 & -2 \\ 4 & -2 \end{pmatrix} (0 - 2) \begin{pmatrix} 4 & -4 \\ -4 & 4 \end{pmatrix} \right]$$

On trouve au total :

$$\widehat{Q}_1 = \frac{1}{4} \left(\begin{pmatrix} 4 & -4 \\ -4 & 4 \end{pmatrix} + \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} + \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} + \begin{pmatrix} 4 & -4 \\ -4 & 4 \end{pmatrix} \right) = \begin{pmatrix} 5/2 & -3/2 \\ -3/2 & 5/2 \end{pmatrix}$$

$$\text{D'où : } |\widehat{Q}_1| = 4 \text{ et : } \widehat{Q}_1^{-1} = \begin{pmatrix} 5/8 & 3/8 \\ 3/8 & 5/8 \end{pmatrix}$$

De même, on trouve :

$$\widehat{\mu}_2 = \begin{pmatrix} 6 \\ 2 \end{pmatrix}, \quad \widehat{Q}_2 = \begin{pmatrix} 5/2 & 3/2 \\ 3/2 & 5/2 \end{pmatrix}, \quad |\widehat{Q}_2| = 4 \text{ et } \widehat{Q}_2^{-1} = \begin{pmatrix} 5/8 & -3/8 \\ -3/8 & 5/8 \end{pmatrix}$$

Puisque les deux classes ont une matrice de covariance de même déterminant, la surface discriminante entre ω_1 et ω_2 est simplement définie par l'équation :

$$(\mathbf{x} - \widehat{\mu}_1)^T \widehat{Q}_1^{-1} (\mathbf{x} - \widehat{\mu}_1) = (\mathbf{x} - \widehat{\mu}_2)^T \widehat{Q}_2^{-1} (\mathbf{x} - \widehat{\mu}_2)$$

$$(x_1 - 2)x_2 - 2) \begin{pmatrix} 5 & 3 \\ 3 & 5 \end{pmatrix} \begin{pmatrix} x_1 - 2 \\ x_2 - 2 \end{pmatrix} = (x_1 - 6)x_2 - 2) \begin{pmatrix} 5 & -3 \\ -3 & 5 \end{pmatrix} \begin{pmatrix} x_1 - 6 \\ x_2 - 2 \end{pmatrix}$$

Après développement, on trouve la surface d'équation :

$$(x_1 - 4)(x_2 + 4/3) = 0$$

Autrement dit, cette surface séparatrice est une hyperbole dégénérée en deux droites qui partage le plan en quatre zones, deux affectées à la classe ω_1 et deux à ω_2 . Cet exemple est représenté sur la figure 14.4.

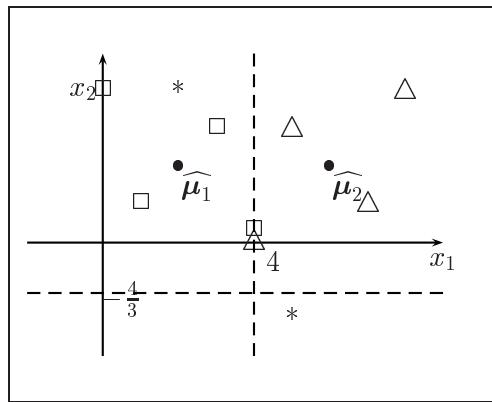


FIG. 14.4 – Deux classes supposées gaussiennes sont représentées chacune par quatre points d'apprentissage notés par \square et \triangle . L'estimation des moyennes $\widehat{\mu}_1$ et $\widehat{\mu}_2$ est indiquée. La surface séparatrice entre les deux classes est l'hyperbole dégénérée composée des deux droites en pointillés. Par conséquent, les deux points * sont classés comme appartenant à la classe \square .

14.2.3 Des hypothèses simplificatrices

Dans un espace de dimension d avec C classes, le nombre de paramètres évalués par les formules 14.13 et 14.14 est d pour chaque moyenne, et $d(d + 1)/2$ pour chaque matrice de covariance, qui est symétrique; soit au total: $(d^2 + 3d)/2$ paramètres par classe et $C(d^2 + 3d)/2$ au total. Si le nombre m_i de points de la classe courante est faible, la précision de ces estimations est mauvaise. On peut donc être amené à faire des hypothèses plus simples; par exemple supposer que toutes les classes ont la même matrice de covariance, ou que celle-ci est diagonale pour chaque classe, ou même les deux à la fois. Voyons les conséquences de chacun de ces biais supplémentaires.

Toutes les classes sont supposées avoir la même matrice de covariance

Dans ce cas, l'estimation des valeurs de la matrice de covariance par la formule 14.14 peut se faire une seule fois sur tous les points de l'ensemble d'apprentissage et non plus classe par classe. Ceci conduit donc à une matrice unique que l'on note \widehat{Q} ; les moyennes sont cependant estimées séparément pour chaque classe par une valeur \widehat{m}_i . Le nombre de paramètres à estimer vaut: $Cd + (d^2 + d)/2$.

La règle de décision peut être interprétée de la façon suivante: pour un point inconnu x , on mesure sa "distance de Mahalanobis" aux vecteurs moyens de chaque classe et on lui attribue la classe la plus proche au sens de cette distance. Celle-ci s'exprime par la formule:

$$D_M(\mathbf{x}, \omega_i) = (\mathbf{x} - \widehat{\boldsymbol{\mu}}_i)^T \widehat{Q} (\mathbf{x} - \widehat{\boldsymbol{\mu}}_i) \quad (14.16)$$

Cette règle de décision revient implicitement à faire une transformation linéaire des coordonnées de chaque point et à prendre la classe dont le centre de gravité est alors le plus proche. Cette transformation « allonge » chaque axe proportionnellement à la valeur propre correspondante dans \widehat{Q} .

Les surfaces séparatrices sont des hyperplans: on se trouve donc ici à une intersection très simple des méthodes bayésiennes et des méthodes de surfaces séparatrices linéaires décrites au chapitre 9.

La classification bayésienne naïve

On suppose ici que chaque classe possède une matrice de covariance diagonale. Cette hypothèse revient à dire que les attributs sont statistiquement décorrélés. Ceci n'étant en général pas vrai, on introduit là un autre type de biais. Cette hypothèse mène à l'estimation de $2Cd$ paramètres et conduit à des séparatrices quadriques de formes particulières (mais pas des hyperplans). Cette hypothèse est souvent appelée la *méthode bayésienne naïve*.

Dans cette simplification, la probabilité d'observer $\mathbf{x}^T = (x_1, \dots, x_d)$ pour un point de n'importe quelle classe ω_i est la probabilité d'observer x_1 pour cette classe, multipliée par celle d'observer x_2 pour cette classe, etc. Donc, par hypothèse:

$$\omega^* = \operatorname{ArgMax}_{i \in \{1, \dots, C\}} P(\omega_i) \prod_{i=1}^d p(x_i | \omega_i)$$

Le problème de trouver la classe

$$\omega^* = \operatorname{ArgMax}_{i \in \{1, \dots, C\}} [P(\omega_i | \mathbf{x})]$$

se ramène donc ici à estimer pour chaque classe la valeur $p(x_1, \dots, x_d | \omega_i)P(\omega_i)$ à partir des données d'apprentissage.

Chaque classe a une matrice de covariance proportionnelle à la matrice identité I

Cette hypothèse impose de plus une isotropie à l'espace de représentation. Elle permet de n'avoir à estimer que $C(d + 1)$ paramètres. Les surfaces qui discriminent chaque classe d'une autre ne sont des hyperplans que si les variances des deux classes sont égales, mais dans ce cas ils sont de plus parallèles aux axes.

14.2.4 Les cas non gaussiens et multigaussiens

On a dit plus haut que le seul cas paramétrique que l'on traitait analytiquement était celui de la distribution gaussienne. Ce n'est pas tout à fait exact : on sait en particulier aussi résoudre le problème en modélisant les classes par des distributions uniformes sur des volumes finis, ou par des distributions exponentielles ; mais ces solutions ne possèdent pas beaucoup d'intérêt pratique.

Une autre cas plus intéressant est celui où l'on suppose que les classes possèdent une distribution de probabilité qui est la somme pondérée de M distributions gaussiennes. Il sera traité dans le chapitre 15 dans le cadre des méthodes d'apprentissage non supervisé.

14.2.5 La prédiction bayésienne de la distribution des paramètres

Au lieu de chercher à identifier une distribution sous-jacente aux données par estimation d'une fonction paramétrée, on peut résoudre directement le problème de la prédiction de la valeur \mathbf{y}_i correspondant à l'observation \mathbf{x}_i . Pour cela, il existe une approche conceptuellement très intéressante et idéalement optimale, même si elle est difficile à mettre en pratique et nécessite de nombreuses approximations.

L'idée essentielle est la suivante. Au lieu de chercher la valeur optimale des paramètres⁶ $\boldsymbol{\theta}$ en maximisant leur vraisemblance sur les données, on décrit ces paramètres comme des distributions de probabilités. Celles-ci sont initialement fixées sous forme d'une distribution *a priori*, puis transformées en distribution *a posteriori* par l'utilisation du théorème de Bayes. Au lieu de chercher une valeur spécifique de $\boldsymbol{\theta}$, on cherche donc ici à trouver la distribution des valeurs s'adaptant le mieux aux données (voir la figure 14.5). La prédiction pour l'événement \mathbf{x} se fait alors en pondérant les valeurs prédites de $\boldsymbol{\theta}$ par la probabilité *a posteriori* correspondante.

Nous avons déjà rencontré cette idée de « vote » des hypothèses dans d'autres contextes, comme celui du boosting d'un algorithme d'apprentissage (11).

Reprendons les notations précédentes, en remarquant que cette fois $\boldsymbol{\theta}$ est un vecteur⁷ aléatoire de densité de probabilité $p(\boldsymbol{\theta})$ connue.

On cherche la densité du vecteur \mathbf{x} étant donné l'échantillon \mathcal{S} :

$$p(\mathbf{x}|\mathcal{S}) = \int p(\mathbf{x}, \boldsymbol{\theta}|\mathcal{S}) d(\boldsymbol{\theta})$$

La formule de Bayes permet d'écrire :

$$p(\mathbf{x}, \boldsymbol{\theta}|\mathcal{S}) = p(\mathbf{x}|\boldsymbol{\theta}, \mathcal{S}) p(\boldsymbol{\theta}|\mathcal{S})$$

Le premier facteur est indépendant de \mathcal{S} puisque nous supposons que la valeur de la densité de \mathbf{x} est entièrement fixée par la valeur du vecteur des paramètres $\boldsymbol{\theta}$.

6. Dans le cas de l'hypothèse d'une distribution gaussienne, ces paramètres sont la moyenne $\boldsymbol{\mu}$ et la matrice de covariance Q .

7. L'ensemble des paramètres est regroupé sous la forme d'un vecteur.

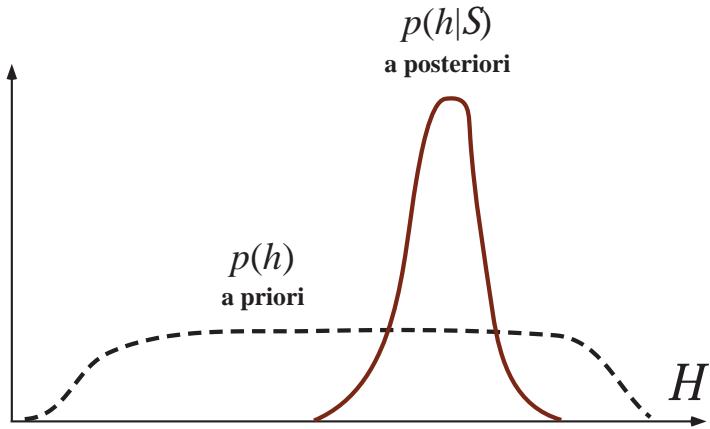


FIG. 14.5 – Illustration schématique de la méthode de prédiction bayésienne. Soit un paramètre θ caractérisant la dépendance fonctionnelle entre les entrées et les sorties. La distribution a priori représente notre connaissance initiale sur la distribution possible de θ qui est typiquement assez lâche. Une fois que les données d'apprentissage ont été prises en compte, la distribution a posteriori calculée par le théorème de Bayes est généralement beaucoup plus focalisée autour d'une ou plusieurs valeurs spécifiques qui sont les plus cohérentes avec les données. Il est alors possible d'utiliser cette distribution pour calculer une prédiction pour une observation nouvelle x_n .

Nous avons donc :

$$p(\mathbf{x}|\mathcal{S}) = \int p(\mathbf{x}|\boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathcal{S}) d\boldsymbol{\theta} \quad (14.17)$$

Ainsi, au lieu de chercher une valeur spécifique de $\boldsymbol{\theta}$, la méthode de prédiction bayésienne calcule une moyenne pondérée sur toutes les valeurs de $\boldsymbol{\theta}$.

Le facteur de pondération $p(\boldsymbol{\theta}|\mathcal{S})$, qui est la distribution *a posteriori* de $\boldsymbol{\theta}$, est déterminé en partant d'une distribution choisie *a priori* $p(\boldsymbol{\theta})$ qui est ensuite mise à jour par utilisation de la règle de Bayes sur l'échantillon d'apprentissage \mathcal{S} . Comme les exemples de cette séquence sont supposés résulter d'un tirage aléatoire suivant une certaine distribution sous-jacente (tirage i.i.d.), on peut écrire :

$$p(\mathcal{S}|\boldsymbol{\theta}) = \prod_{i=1}^m p(\mathbf{x}_i|\boldsymbol{\theta})$$

en utilisant à nouveau le théorème de Bayes :

$$p(\boldsymbol{\theta}|\mathcal{S}) = \frac{p(\mathcal{S}|\boldsymbol{\theta}) p(\boldsymbol{\theta})}{p(\mathcal{S})} = \frac{p(\boldsymbol{\theta})}{p(\mathcal{S})} \prod_{i=1}^m p(\mathbf{x}_i|\boldsymbol{\theta}) \quad (14.18)$$

où le facteur de normalisation $p(\mathcal{S})$ est donné par :

$$p(\mathcal{S}) = \int p(\boldsymbol{\theta}') \prod_{i=1}^m p(\mathbf{x}_i|\boldsymbol{\theta}') d\boldsymbol{\theta}'$$

ce qui assure que $\int p(\boldsymbol{\theta}|\mathcal{S}) d\boldsymbol{\theta} = 1$.

L'évaluation d'une intégrale comme celle de l'équation 14.17 n'est possible de manière analytique que pour une classe de fonctions de densité pour lesquelles la densité *a posteriori* $p(\boldsymbol{\theta}|\mathcal{S})$ a la même forme que la densité *a priori* $p(\boldsymbol{\theta})$. Dans ce cas particulier, on parle de *densités autoreproductibles* [DH73]. L'exemple le plus commun de telles densités est celui de la distribution normale (gaussienne).

Illustration avec une loi normale unidimensionnelle

Supposons que les observations \mathbf{x} soient décrites par une mesure unidimensionnelle qui suit une loi normale de moyenne inconnue $\boldsymbol{\mu}$ et de variance σ connue.

L'approche de la prédiction bayésienne nous dicte de chercher la densité de probabilité de la variable $\boldsymbol{\mu}$ aléatoire en fonction des données d'apprentissage \mathcal{S} . Nous supposons que le paramètre $\boldsymbol{\mu}$ suit également une loi normale de moyenne $\boldsymbol{\mu}_0$ et de variance σ_0^2 . Pour exprimer notre ignorance *a priori* sur la valeur de $\boldsymbol{\mu}$, nous prenons une grande valeur pour la variance σ_0^2 .

$$p_0(\boldsymbol{\mu}) = \frac{1}{(2\pi\sigma_0^2)^{\frac{1}{2}}} \exp -\frac{(\boldsymbol{\mu} - \boldsymbol{\mu}_0)^2}{2\sigma_0^2}$$

La donnée d'une séquence d'apprentissage \mathcal{S} permet de réviser cette densité de probabilité en utilisant le théorème de Bayes suivant l'équation (14.18) :

$$p(\boldsymbol{\mu}|\mathcal{S}) = \frac{p_0(\boldsymbol{\mu})}{p(\mathcal{S})} \prod_{i=1}^m p(\mathbf{x}_i|\boldsymbol{\mu})$$

En utilisant le fait que :

$$p(\mathbf{x}|\boldsymbol{\mu}) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp -\frac{(\mathbf{x} - \boldsymbol{\mu})^2}{2\sigma^2}$$

il est facile de montrer que la distribution *a posteriori* $p(\boldsymbol{\mu}|\mathcal{S})$ est également normale avec :

$$\boldsymbol{\mu} = \frac{m\sigma_0^2}{m\sigma_0^2 + \sigma^2} \bar{\mathbf{x}} + \frac{\sigma^2}{m\sigma_0^2 + \sigma^2} \boldsymbol{\mu}_0 \quad (14.19)$$

$$\frac{1}{\sigma^2} = \frac{m}{m\sigma_0^2 + \sigma^2} \quad (14.20)$$

où $\bar{\mathbf{x}}$ est la moyenne : $\bar{\mathbf{x}} = \frac{1}{m} \sum_{i=1}^m \mathbf{x}_i$.

Ces équations montrent qu'au fur et à mesure que le nombre m de données augmente, la moyenne $\boldsymbol{\mu}$ de la distribution *a posteriori* approche la moyenne de l'échantillon d'apprentissage $\bar{\mathbf{x}}$. De même, l'écart type σ décroît vers zéro.

Ainsi l'approche de la prédiction bayésienne calcule une moyenne pondérée sur toutes les valeurs de $\boldsymbol{\theta}$ au lieu de choisir une valeur spécifique. Cependant si la densité *a posteriori* $p(\boldsymbol{\theta}|\mathcal{S})$ présente un pic étroit centré sur une valeur $\hat{\boldsymbol{\theta}}$, alors $p(\boldsymbol{\theta}|\mathcal{S}) \approx p(h|\hat{\boldsymbol{\theta}})$, et nous retrouvons le résultat donné par la méthode du maximum de vraisemblance. Cela arrive généralement pour les échantillons d'apprentissage de grande taille.

Bien que cela ne soit pas le sujet de ce chapitre, il est utile de noter dès à présent que le principe du maximum de vraisemblance et l'apprentissage bayésien ne se prêtent pas aux mêmes méthodes de calcul. Le premier se traite comme un problème d'optimisation : il faut chercher le minimum d'une fonction d'erreur. En revanche, dans le second, l'essentiel du calcul implique une intégration sur des espaces de grandes dimensions. Dans ce dernier cas, les méthodes classiques d'intégration ne conviennent pas et il faut se tourner vers des méthodes approchées, par exemple les méthodes de Monte-Carlo (voir le chapitre 3).

14.3 L'apprentissage bayésien non paramétrique

14.3.1 Généralités : le problème de l'estimation locale d'une densité

Les méthodes non paramétriques traitent de l'estimation d'une densité de probabilités pour laquelle aucune régularité fonctionnelle n'est supposée *a priori*. Ces méthodes reposent cependant sur l'hypothèse fondamentale que les distributions ou fonctions recherchées sont localement régulières.

Soit une densité de probabilité inconnue $p(\mathbf{x})$. La probabilité Q pour qu'une forme \mathbf{x} issue de cette distribution soit observée dans la région $\mathcal{R} \in \mathcal{X}$ est :

$$Q = \int_{\mathcal{R}} p(\mathbf{u}) d(\mathbf{u})$$

L'annexe 18.4 explique comment on obtient une bonne estimation de Q à partir de la moyenne des points observés dans la région \mathcal{R} :

$$Q \approx k/m. \quad (14.21)$$

Par ailleurs, en faisant l'hypothèse que la densité cherchée $p(\mathbf{x})$ est continue et ne varie pas significativement dans la région \mathcal{R} , on peut faire l'approximation :

$$Q = \int_{\mathcal{R}} p(\mathbf{u}) d\mathbf{u} \approx p(\mathbf{x}) V \quad (14.22)$$

où V est le volume de la région \mathcal{R} .

De (14.21) et (14.22) on déduit :

$$p(\mathbf{x}) \approx \frac{k}{mV} \quad (14.23)$$

Application à l'apprentissage d'une règle de classification

Dans le cas où l'on cherche à apprendre une règle de classification, la méthode bayésienne consiste à estimer en un point \mathbf{x} donné la densité de probabilité de chaque classe afin de choisir celle qui possède la valeur la plus grande.

Nous omettons dans ce qui suit l'indice correspondant au numéro de classe, puisque le problème est le même pour chaque classe. Comme ce n'est qu'après l'estimation séparée pour chaque classe qu'on les compare, on peut faire comme s'il n'y avait qu'une seule densité de probabilité à estimer.

Néanmoins nous allons indexer les termes par m , la taille de l'échantillon : on verra que cette précision est nécessaire quand on étudie les propriétés de convergence.

On suppose donc être en possession de m points de \mathbb{R}^d obtenus par tirages indépendants selon une densité qui caractérise la classe ω . Comment estimer $p(\mathbf{x} | \omega)$ au point \mathbf{x} à partir d'un ensemble d'apprentissage ? Le principe vient d'être expliqué : on définit autour de \mathbf{x} une certaine région \mathcal{R}_m (en pratique, une hypersphère ou un hypercube) et on compte le nombre k_m de points de l'échantillon d'apprentissage qui sont inclus dans ce volume (figure 14.6).

On a vu que l'estimateur de $p(\mathbf{x} | \omega)$ pour un échantillon de taille m se définit par :

$$\widehat{p}_m(\mathbf{x} | \omega) = \frac{k_m/m}{V_m} \quad (14.24)$$

où V_m est le volume de la région \mathcal{R}_m considérée.

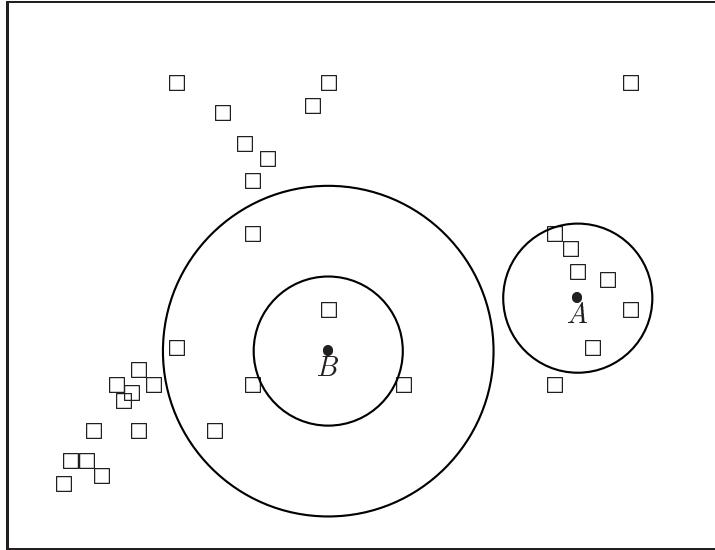


FIG. 14.6 – Les points \square sont des tirages indépendants selon une certaine distribution dans le plan \mathbb{R}^2 , dont la densité est plus forte au point A qu’au point B . En effet, pour le même volume autour du point A et du point B , k_m vaut respectivement 6 et 1. Pour avoir $k_m = 6$ autour du point B , il faut augmenter le volume.

On peut démontrer (voir l’annexe 18.4) que, quand m augmente, cet estimateur converge vers la valeur cherchée $p(\mathbf{x} | \omega)$, quand les conditions suivantes sont remplies :

$$\lim_{m \rightarrow \infty} V_m = 0$$

$$\lim_{m \rightarrow \infty} k_m = \infty$$

$$\lim_{m \rightarrow \infty} (k_m/m) = 0$$

Il y a en pratique deux solutions pour remplir ces conditions :

- Soit définir V_m à partir d’une région \mathcal{R}_0 de forme et de volume V_0 fixés : par exemple un hypercube de côté unité, mais on verra que c’est loin d’être le seul cas possible. On prend alors :

$$V_m = V_0/f(m)$$

où f est une fonction croissante de m . Ceci conduit aux méthodes des fonctions noyau, en particulier aux *fenêtres de Parzen*.

- Soit fixer le nombre k_m , se donner une famille de volumes paramétrée par une variable (par exemple les hypersphères centrées en \mathbf{x} , de rayon variable) et ajuster cette variable pour que le volume contienne exactement k_m points de l’ensemble d’apprentissage. Cette technique d’estimation est connue sous le nom de méthode des *k-plus proches voisins*. Utilisée dans le problème de la classification, elle se traduit par un algorithme qui ne nécessite pas l’estimation explicite de la densité de chaque classe au point à classer, mais en permet plus simplement un choix direct.

14.3.2 Les fonctions noyau et les fenêtres de Parzen

Une fonction noyau (*kernel*) K est une fonction bornée sur \mathcal{X} d’intégrale égale à 1. On suppose en général que K présente un pic centré en 0. Par conséquent, $K(\mathbf{x}_i - \mathbf{x}_j)$ détermine

une mesure de proximité entre les points \mathbf{x}_i et \mathbf{x}_j . On impose aussi en général que cette fonction soit symétrique : $K(-\mathbf{x}) = -K(\mathbf{x})$. Dans cette perspective, l'estimation locale de la densité $p(\mathbf{x})$ est prise comme une somme pondérée des exemples \mathbf{x}_j pondérée par leur distance à \mathbf{x} :

$$\hat{p}(\mathbf{x}) = \frac{1}{m} \sum_{j=1}^m K(\mathbf{x} - \mathbf{x}_j)$$

ce qui peut aussi être interprété comme une moyenne des fonctions noyau centrées sur chaque exemple.

Pour une classe ω_k donnée, on a :

$$\hat{p}(k|\mathbf{x}) = \frac{\hat{p}(\omega_k) \hat{p}_k(\mathbf{x})}{\sum_{i=1}^C \hat{p}(\omega_i) \hat{p}_i(\mathbf{x})} = \frac{\frac{\hat{p}(\omega_k)}{m_k} \sum_{\mathbf{x}_i \in \omega_k} K(\mathbf{x} - \mathbf{x}_i)}{\sum_{i=1}^C \frac{\hat{p}(\omega_i)}{n_i} K(\mathbf{x} - \mathbf{x}_i)}$$

Si les probabilités des classes sont estimées par le rapport m_k/m où m_k est le nombre d'exemples de la classe ω_k et m le nombre total d'exemples, on en déduit :

$$\hat{p}(k|\mathbf{x}) = \frac{\sum_{\mathbf{x}_i \in \omega_k} K(\mathbf{x} - \mathbf{x}_i)}{\sum_{i=1}^m K(\mathbf{x} - \mathbf{x}_i)} \quad (14.25)$$

ce qui revient à prendre la proportion pondérée d'exemples autour de \mathbf{x} de la classe ω_k .

Une difficulté de ces méthodes est le choix de la fonction K . Une autre difficulté provient de leur médiocre capacité à être utilisées dans des espaces de représentation \mathcal{X} de grande dimension. En effet, les estimations sont fondées sur la détermination d'un volume dans l'espace des données. Or, dans les espaces de grande dimension, un volume qui couvre suffisamment de données n'est plus valide pour une estimation locale, car son rayon tend à devenir grand par rapport à l'intervalle des valeurs possibles pour les données.

Le paragraphe suivant explore ces méthodes d'estimation par voisinage dans \mathcal{X} . Les cas des fonctions noyau définies comme des hypercubes ou des distributions gaussiennes y sont en particulier traités. La méthode qui en résulte s'appelle la méthode des fenêtres de Parzen.

14.3.2.1 Les fenêtres de Parzen : le cas élémentaire

Commençons en définissant le volume élémentaire \mathcal{R}_m comme un hypercube de côté h_m centré en \mathbf{x} . On a dans ce cas :

$$V_m = h_m^d$$

$$\widehat{p}_m(\mathbf{x}) = \frac{1}{m V_m} \sum_{i=1}^m \frac{\phi(\mathbf{x} - \mathbf{x}_i)}{h_m}$$

où ϕ est la fonction caractéristique de l'hypercube unité :

$$\phi(\mathbf{x}_i) = 1 \text{ si } \mathbf{x} \in [-1/2, 1/2]$$

$$\phi(\mathbf{x}_i) = 0 \text{ sinon}$$

Les conditions de convergence citées plus haut au paragraphe 14.3.1 sont remplies par exemple en prenant, pour une valeur h_0 fixée :

$$h_m = \frac{h_0}{m}$$

La formule 14.24 définit par conséquent l'estimateur de $p(\mathbf{x} | \omega)$. k_m est le nombre de points d'apprentissage de la classe ω inclus dans l'hypercube centré en \mathbf{x} de côté h_m . En pratique, pour estimer $p_m(\mathbf{x} | \omega)$, il ne reste qu'à fixer la valeur h_0 : on en déduit la valeur de h_m , puis celle de $\widehat{p}_m(\mathbf{x} | \omega)$ par comparaison des coordonnées des points de l'ensemble d'apprentissage avec celles de \mathbf{x} .

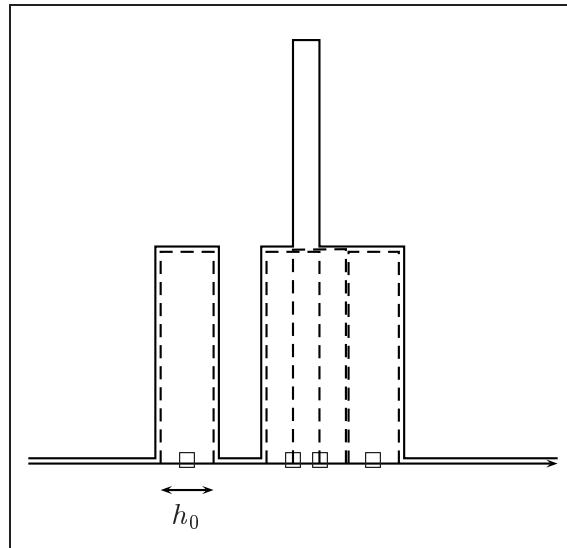


FIG. 14.7 – *Estimation de densité par la méthode des fenêtres de Parzen. Il y a quatre points d'apprentissage, dans un espace à une dimension. les hypercubes sont des segments de largeur h_0 centrés sur les points d'apprentissage. La densité (en trait plein) est calculée comme la somme des fenêtres centrées sur chaque point. Ici, cette fenêtre est étroite (h_0 est petit) : la densité résultante est peu lisse. La surface sous la courbe en trait plein est égale à 1.*

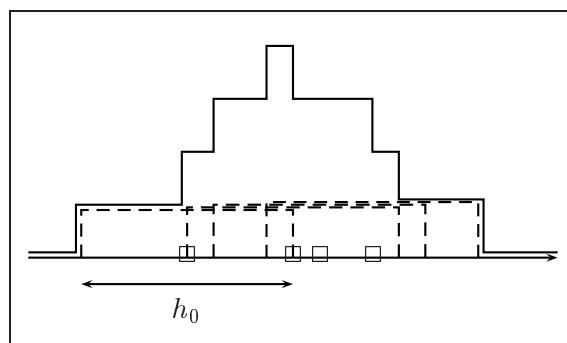


FIG. 14.8 – *La même estimation pour h_0 plus grand : la densité est estimée de manière plus lisse.*

Le choix de h_0 influe sur le résultat de la façon suivante : si cette valeur est choisie petite, la probabilité estimée que le point \mathbf{x} ait été engendré par le processus ω est nulle partout, sauf au voisinage immédiat des points de l'ensemble d'apprentissage; on a donc dans ce cas modélisé $p(\omega)$ comme un « peigne ». Si elle est choisie grande, $p(\omega)$ est en revanche modélisée de manière « lisse » (figures 14.7 et 14.8).

14.3.2.2 Généralisation à des fonctions noyau gaussiennes

La technique précédente a l'avantage de se réduire à un algorithme simple, mais présente l'inconvénient de la sensibilité du choix de la valeur h_0 . D'autre part, dès que l'on s'éloigne des points d'apprentissage, la densité est estimée comme nulle. On peut remédier à ce problème en s'arrangeant pour que l'appartenance au volume élémentaire V_m autour du point \mathbf{x} ne soit plus une fonction caractéristique (à valeur binaire) d'appartenance à un volume, mais une probabilité.

C'est ce qui a été présenté ci-dessus en introduction : la densité est estimée comme la somme de noyaux, qui sont donc des densités élémentaires centrées sur les points d'apprentissage. Souvent, ces noyaux sont des distributions gaussiennes (voir la figure 14.9). Le résultat est plus ou moins lisse selon la valeur de la variance, mais il n'y a plus de point de l'espace où la densité soit estimée comme nulle. Sans entrer dans les détails, on peut voir intuitivement cette généralisation

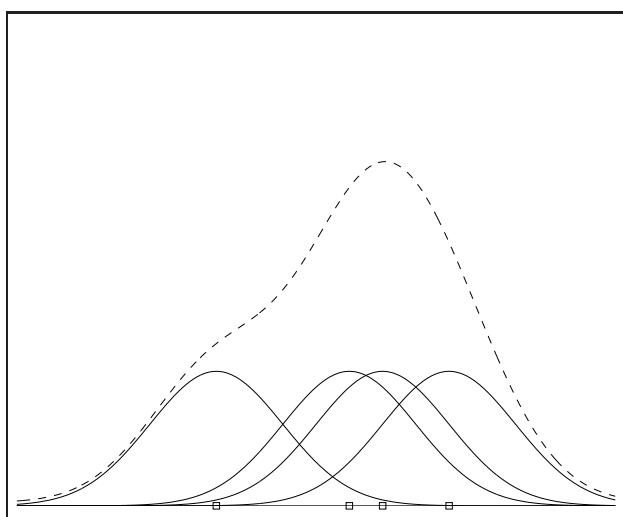


FIG. 14.9 – Fenêtres de Parzen : estimation avec un noyau gaussien.

de la façon suivante: pour estimer la valeur $p_m(\mathbf{x} | \omega)$, on additionne sur les m points de l'ensemble d'apprentissage des valeurs entre 0 et 1 qui sont fonction de la distance entre \mathbf{x} et le point courant, avant de diviser cette somme par un coefficient normalisateur. Cette fonction n'est donc plus binaire comme dans le cas précédent, mais calculée de façon continue. On doit évidemment la choisir (ainsi que la normalisation finale) de façon que $p_m(\mathbf{x} | \omega)$ soit effectivement une estimation de densité de probabilité. La figure 14.9 montre l'estimation réalisée à partir des mêmes points que dans les figures 14.8 et 14.7 pour des noyaux gaussiens.

14.3.3 Les k -plus proches voisins (k -ppv)

L'un des problèmes avec les méthodes par fonctions noyau provient de ce que leur taille est fixe. Si celle-ci est trop grande, l'approximation peut être trop « lissée » par rapport à la réalité. Si elle trop petite, l'estimation dans des régions de faible densité peut être nulle ou très approximative. Il faudrait donc que cette taille soit fonction de la position dans l'espace \mathcal{X} . C'est ce que réalise la méthode par plus proches voisins.

Dans celle-ci, le nombre k de points dans la région autour de \mathbf{x} est fixé et on fait au contraire varier le volume V . On considère donc une hypersphère (en général on utilise une distance euclidienne) centrée en \mathbf{x} et on fait varier le rayon jusqu'à ce qu'elle contienne k points. L'estimation

de la densité est alors donnée par le rapport k/mV où m est le nombre total de points dans l'échantillon de données.

D'un certain côté, cela revient à choisir une fonction noyau simple, constante sur l'hyper-sphère contenant les k points et nulle ailleurs. Mais cela permet de passer directement à une règle de décision: on classe une forme inconnue \boldsymbol{x} en prenant la classe qui est majoritaire dans les k points d'apprentissage les plus proches. Cette règle est appelée règle des k -plus proches voisins (*k -nearest-neighbour classification rule*) où k est le nombre de voisins considérés.

Dans le cas où $k = 1$, la règle s'appelle la *règle de classification du plus proche voisin*. Elle assigne à \boldsymbol{x} simplement la même étiquette que le point d'apprentissage le plus proche. Dans ce cas, les frontières de décision dans l'espace \mathcal{X} prennent la forme d'un pavage convexe (voir la figure 14.12). Il est remarquable que cette règle extrêmement simple possède un comportement asymptotique excellent vis-à-vis du risque minimal de Bayes, comme on le verra au paragraphe 14.3.3.2.

Il est conseillé de se reporter à [DH73] et [DK82] pour une étude approfondie des méthodes de plus proches voisins. Le lecteur francophone lira avec profit [CL96].

14.3.3.1 Le principe

La règle de décision par k -ppv est facile à illustrer, comme sur la figure 14.10.

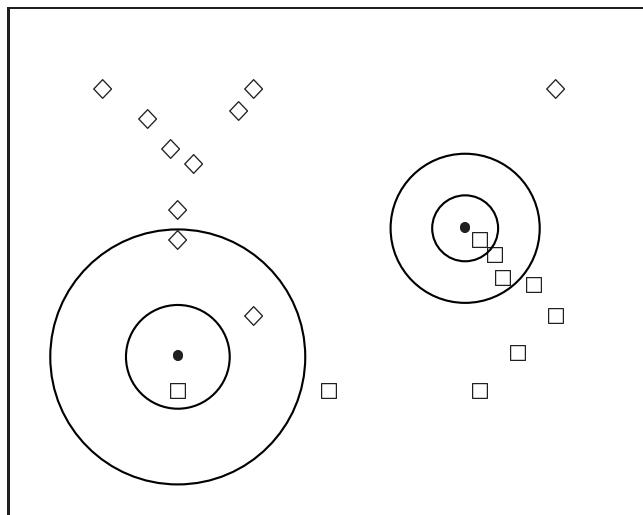


FIG. 14.10 – *Décision par 1-ppv et 3-ppv dans un ensemble d'exemples appartenant à deux classes.*

On y a représenté un problème à deux classes : les points à classer sont notés \bullet et les points alentour sont les données d'apprentissage, appartenant soit à la classe notée \square , soit à celle notée \diamond . On cherche, au sens de la métrique choisie pour le problème (sur ce dessin, euclidienne), les k -plus proches voisins des points \boldsymbol{x} ; pour $k = 1$, dans les deux cas, c'est un des points notés \square . On affecte donc aux deux points \bullet à classe \diamond . Pour $k = 3$, le voisinage du premier point \bullet compte deux points \diamond et un point \square : c'est la classe \diamond qui est majoritaire, et ce point est classé comme appartenant à la classe \diamond . Pour l'autre point, la décision pour $k = 3$ confirme l'appartenance à la classe \square .

La figure 14.11 représente la même opération pour un problème à trois classes. Pour $k = 1$, les points \bullet sont classés comme \square ; pour $k = 3$, la règle de décision produit une ambiguïté pour le premier point: on ne peut pas se décider entre les trois classes.

L'algorithme de la méthode est donné en 14.1.

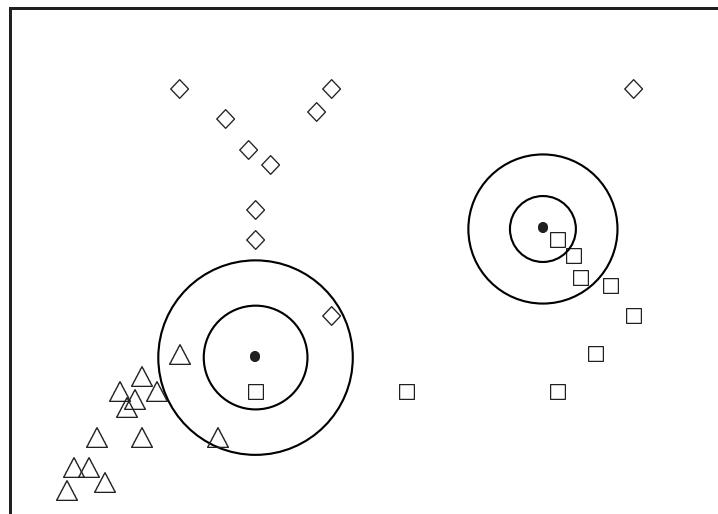


FIG. 14.11 – Décision par 1-ppv et 3-ppv dans un ensemble d'exemples appartenant à trois classes.

Algorithme 14.1 Algorithme des k -plus proches voisins

Début

On cherche à classer le point \mathbf{x}

pour chaque exemple (\mathbf{y}, ω) de l'ensemble d'apprentissage **faire**

 calculer la distance $D(\mathbf{y}, \mathbf{x})$ entre \mathbf{y} et \mathbf{x}

fin pour

Dans les k points les plus proches de \mathbf{x}

 compter le nombre d'occurrences de chaque classe

Attribuer à \mathbf{x} la classe qui apparaît le plus souvent

Fin

14.3.3.2 La validité bayésienne

Quelle est la validité de cette règle en apparence naïve ? Elle est conforme aux règles de l'estimation bayésienne définies au paragraphe 14.3.1, sous l'hypothèse que les probabilités *a priori* des classes sont bien estimées par leur proportion d'échantillons d'apprentissage.

La règle des k -ppv fait implicitement une estimation comparative de toutes les densités de probabilités des classes apparaissant dans le voisinage de \mathbf{x} et choisit simplement la plus probable : elle approxime donc la décision bayésienne.

Pour s'en convaincre, il suffit de supposer que les m points de l'ensemble d'apprentissage comportent m_i points de la classe ω_i et que sur les k -plus proches voisins de \mathbf{x} , il y a k_{m_i} points de cette classe. On a, d'après l'équation 14.24 :

$$\widehat{p}_m(\mathbf{x} | \omega_i) = \frac{k_{m_i}/m_i}{V_m}$$

On fait maintenant l'hypothèse que m_i/m est un estimateur de $P(\omega_i)$, la probabilité *a priori* de la classe de rang i . On peut donc noter : $m_i/m = \widehat{P}_m(\omega_i)$.

On en déduit :

$$k_{m_i} = m \cdot V_m \cdot \widehat{p}_m(\mathbf{x} \mid \omega_i) \cdot \widehat{P}_m(\omega_i)$$

Par conséquent, la classe qui a le plus de points d'apprentissage dans les k_m (celle pour laquelle la valeur k_{m_i} est maximale) est aussi celle qui maximise la valeur $p_m(\mathbf{x} \mid \omega_i) \cdot \widehat{P}_m(\omega_i)$ qui est égale, par la règle de Bayes, à : $\widehat{P}_m(\omega_i \mid \mathbf{x}) \cdot p(\mathbf{x})$. Cette classe est donc celle qui maximise la valeur $\widehat{P}_m(\omega_i \mid \mathbf{x})$. Son choix approxime par conséquent la règle de classification bayésienne.

Rappelons que tout ce calcul ne vaut que si m_i/m est un estimateur de $P(\omega_i)$. Il faut donc n'appliquer la règle des k -ppv qu'après s'être assuré de la validité de cette hypothèse.

14.3.3.3 Quelques propriétés de convergence

Il est assez facile de démontrer que la probabilité d'erreur R_{k-ppv} de la règle des $k-ppv$ converge vers le risque bayésien R_B quand m , le nombre total d'échantillons, croît vers l'infini, et ceci pour tout k . Cette propriété est démontrée en annexe 18.6 pour $k = 1$.

On a de plus les propriétés suivantes, dans le cas de deux classes, toujours à la limite sur m :

$$R_B \leq R_{k-ppv} \leq R_{(k-1)-ppv} \cdots \leq R_{1-ppv} \leq 2R_B \quad (14.26)$$

avec :

$$R_{k-ppv} \leq R_B + R_{1-ppv} \sqrt{\frac{2}{\pi k}} \quad (14.27)$$

et pour un nombre quelconque C de classes :

$$R_{1-ppv} \leq R_B \left(2 - \frac{C}{C-1} R_B \right) \quad (14.28)$$

Ces formules valident donc l'intuition que l'augmentation de k améliore l'estimation réalisée ; en même temps, elles prouvent que la règle simple du plus proche voisin (1-ppv) est asymptotiquement efficace. On résume souvent plaisamment la formule 14.26 par l'expression : « la moitié de l'information sur la classification optimale d'un point inconnu est disponible dans son seul plus proche voisin ».

Mais les formules ci-dessus ne sont valables que pour m assez grand, ce qui est une remarque pratique importante. Pour plus de détails, on peut se reporter avec profit à [Rip96] (pp.192-197).

14.3.3.4 Considérations pratiques

Bien sûr, dans la problématique de l'apprentissage, le nombre m est fini ; il faut alors trouver un compromis entre une valeur faible de k , qui semble moins favorable selon les formules ci-dessus, et une valeur exagérément grande⁸. Diverses considérations théoriques et expérimentales mènent à l'heuristique suivante: choisir k autour de $\sqrt{m/C}$ où m/C est le nombre moyen de points d'apprentissage par classe. On remarquera que d , la dimension de l'espace de représentation, n'apparaît pas dans cette formule.

8. Prendre $k = m$ mène à au résultat suivant : tous les points seront classés comme appartenant à la classe la plus nombreuse dans l'ensemble d'apprentissage ; seule l'estimation *a priori* des classes compte alors.

Un autre problème pratique est : quelle décision prendre en cas d'égalité ? On peut augmenter k de 1 pour trancher le dilemme, mais s'il y a plus de deux classes, l'ambiguïté peut subsister. Une autre solution consiste à tirer au hasard la classe à attribuer au point ambigu ; son analyse montre qu'elle n'est pas mauvaise.

Enfin, un grand nombre d'auteurs ont proposé des variantes de la règle du k -ppv ; par exemple, au lieu de compter simplement les points de chaque classe parmi les k (ce que l'on peut traduire par : les faire voter avec une voix chacun), on a pensé pondérer ces votes par la distance au point x , qui est de toute façon calculée. On est dans ce cas dans des méthodes intermédiaires entre les k -plus proches voisins et les fenêtres de Parzen.

14.3.3.5 Les surfaces séparatrices de la règle de décision k -ppv

Il est courant d'appeler *zone de Voronoï* d'un exemple le lieu des points de \mathbb{R}^d qui sont plus proches de cet exemple que de tout autre exemple.

Des considérations géométriques permettent de prouver que la zone de Voronoï d'un exemple est l'intersection de $m - 1$ demi-espaces, définis par les hyperplans médiateurs entre cet exemple et tous les autres. La zone de Voronoï d'un exemple est donc un volume convexe (pour $d = 2$, c'est un polygone convexe) et la frontière entre deux zones de Voronoï est un « polygone » en dimension $d - 1$.

Pour $k = 1$, la surface séparatrice entre deux classes est la surface séparatrice entre les deux volumes obtenus en faisant l'union des surfaces de Voronoï des exemples de chaque classe (voir la figure 14.12). On peut montrer que pour $k > 1$, les séparatrices sont encore des hyperplans par morceaux.

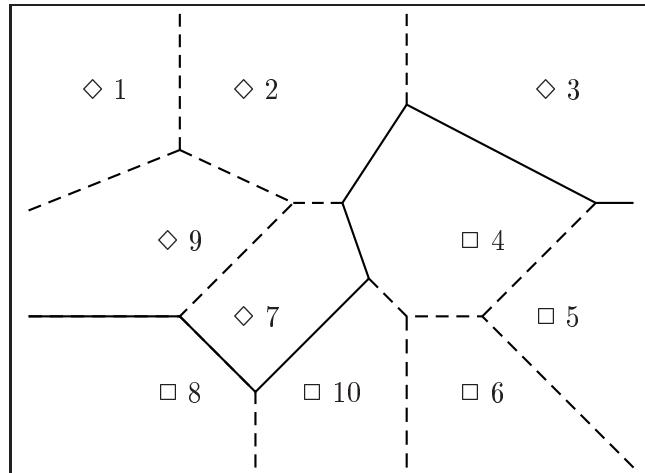


FIG. 14.12 – Un ensemble de points appartenant à deux classes et leurs zones de Voronoï. La séparatrice entre les deux classes par la règle de décision $1 - \text{ppv}$ est en trait plein.

14.3.3.6 Algorithmique avancée pour les k -ppv

L'algorithme de base de la décision par k -ppv consiste à calculer les m distances du point x à classer aux points d'apprentissage, et à trouver au fur et à mesure les k plus faibles distances parmi les m , pour choisir la classe majoritaire dans les k points d'apprentissage ainsi sélectionnés. C'est à chaque fois un calcul en $\mathcal{O}(m \times d)$ qu'il faut effectuer pour prendre une décision. Ceci est à comparer à la quantité de calculs que requiert une décision quand l'ensemble

d'apprentissage a été « compilé » par apprentissage paramétrique, par exemple sous la forme de distributions gaussiennes explicites : au plus en $\mathcal{O}(Cd^2)$. En général, la comparaison n'est pas à l'avantage de la règle des k -ppv, en tout cas dès que l'on dispose d'un ensemble d'apprentissage un peu conséquent, comme il faut le souhaiter. C'est pourquoi une algorithmique particulière a été développée, visant soit à réduire l'ensemble d'apprentissage (méthodes de *nettoyage* et de *condensation*) sans changer le résultat des futures décisions par k -ppv, soit à l'organiser sous des structures de données permettant d'accélérer la décision (méthodes *rapides* de k -ppv).

Nettoyage et condensation de l'ensemble d'apprentissage

Le nettoyage d'un ensemble d'apprentissage est une technique très générale, reliée mathématiquement aux méthodes de validation statistique d'un apprentissage. On la présente ici comme une technique inséparable de l'algorithme de condensation : l'usage recommande en effet de ne pas utiliser le second sans le premier.

Algorithme 14.2 Algorithme de nettoyage

Début

Diviser aléatoirement l'ensemble d'apprentissage en deux sous-ensembles S_1 et S_2

tant que la stabilisation de S_1 et S_2 n'est pas réalisée **faire**

 Classer tous les points de S_1 sur S_2 par la règle du 1-ppv

 Eliminer de S_1 tous les points dont la classe

 n'est pas la même que celle de leur plus proche voisin dans S_2

 Classer tous les points de S_2 sur le nouveau S_1 par la règle du 1-ppv

 Eliminer de S_2 tous les points dont la classe

 n'est pas la même que celle de leur plus proche voisin dans S_1

fin tant que

L'ensemble d'apprentissage nettoyé est composé de $S_1 \cup S_2$

Fin

Algorithme 14.3 Algorithme de condensation

Début

Ordonner les m exemples d'apprentissage de \mathbf{x}_1 à \mathbf{x}_m

Initialiser S par \mathbf{x}_1 et G par \mathbf{x}_2 à \mathbf{x}_m

tant que S et G ne sont pas stabilisés **faire**

pour Chaque point g_i de G **faire**

si Le $1 - ppv$ de g_i dans S n'a pas la même classe que g_i **alors**

 Enlever g_i de G et le mettre dans S

fin si

fin pour

fin tant que

L'ensemble d'apprentissage condensé est S

Fin

L'algorithme de nettoyage n'est pas très utile en soi, dans la mesure où il réduit généralement

ment assez peu la taille de l'ensemble d'apprentissage; mais il constitue un prétraitement très efficace à la condensation. En effet, la même idée guide ces deux algorithmes: éliminer les points inutiles du point de vue de la décision par plus proche voisin. Mais ils ne s'intéressent pas à l'élimination des mêmes points: la condensation ne garde que les points proches de la frontière des classes; le nettoyage élimine les points isolés et par conséquent définit ces frontières comme plus simples. Sur les exemples des figures 14.14, 14.15, 14.16 et 14.17, on voit quatre versions du même ensemble d'apprentissage (à deux dimensions et à deux classes): son état original, le résultat de son nettoyage, le résultat de sa condensation et celui de la condensation après le nettoyage.

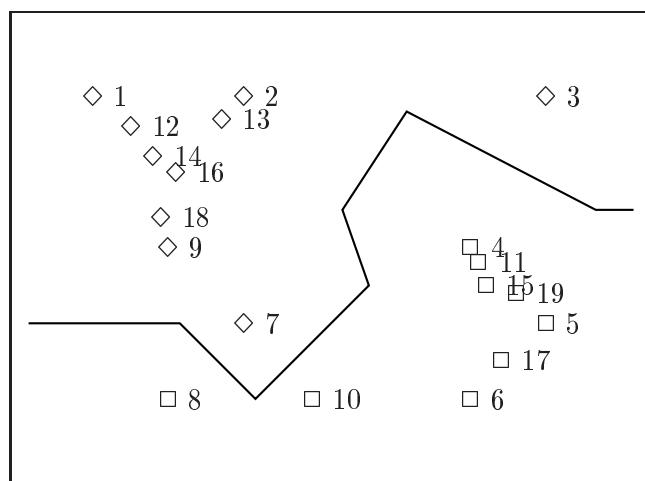


FIG. 14.13 – *Un ensemble d'exemples appartenant à deux classes. La surface séparatrice par la règle du 1-ppv est une ligne brisée composée de segments de médiatrices entre couples de points de classe différente.*

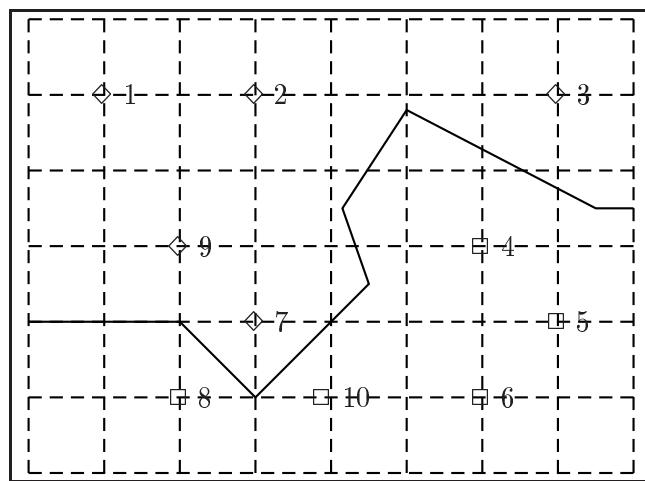


FIG. 14.14 – *Le même ensemble d'exemples, simplifié.*

Ces méthodes sont valides au sens où elles ne changent pas le résultat de la classification par k -ppv quand le nombre m d'exemples augmente indéfiniment. En pratique, il a été constaté que leur efficacité diminue considérablement quand la dimension d de l'espace de représentation

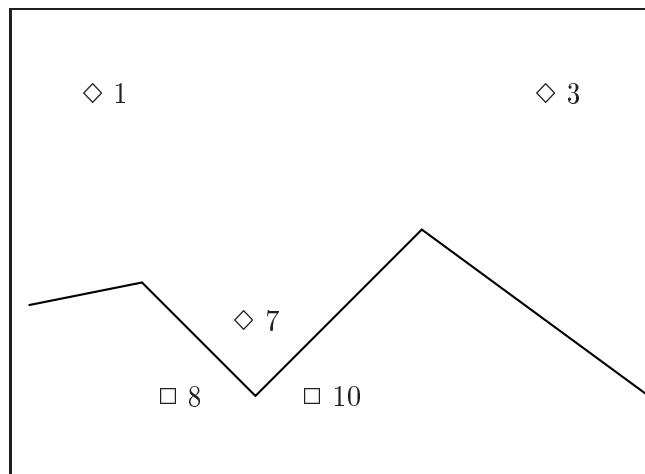


FIG. 14.15 – L’ensemble simplifié après condensation et la nouvelle surface séparatrice par la règle du 1-ppv.

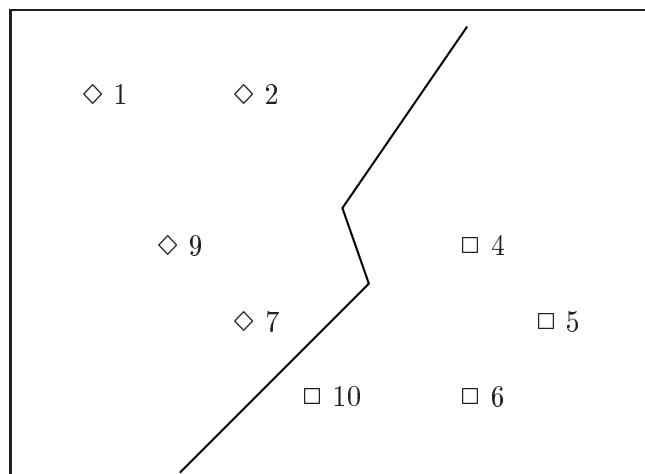


FIG. 14.16 – L’ensemble simplifié après nettoyage et la nouvelle surface séparatrice par la règle du 1-ppv.

augmente, même si le nombre des points de l’ensemble d’apprentissage augmente en proportion.

Parcours accéléré de l’ensemble d’apprentissage

Les méthodes précédentes ont pour but de réduire la taille de l’ensemble d’apprentissage et donc en proportion le temps nécessaire au classement d’un point inconnu. Il existe d’autres techniques pour réduire ce temps tout en préservant exactement l’ensemble d’apprentissage. La plupart sont basées sur l’inégalité triangulaire de la distance euclidienne Δ . On suppose pour cette méthode que l’on a calculé par avance toutes les distances entre les points d’apprentissage.

L’idée est alors la suivante : soit x le point à classer ; on est en train de parcourir l’ensemble des points d’apprentissage et le plus proche voisin de x est pour le moment un certain point d’apprentissage y à la distance $\Delta(x, y) = \delta$. Soit z le point suivant dans l’ensemble d’appren-

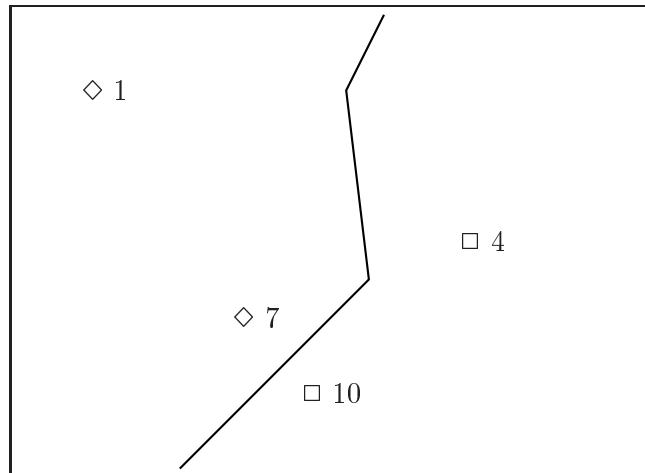
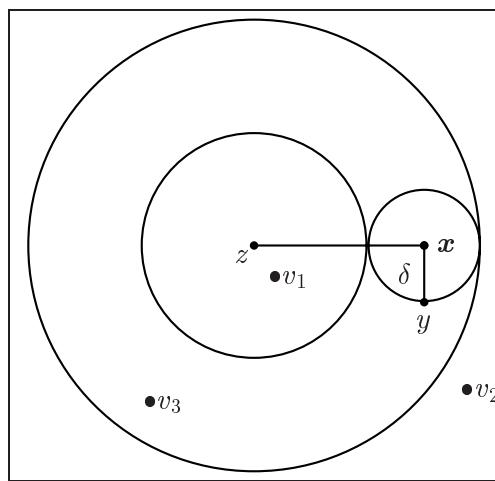


FIG. 14.17 – L'ensemble simplifié après nettoyage puis condensation.

FIG. 14.18 – Méthode accélérée de recherche du plus proche voisin. Le ppv courant de \mathbf{x} est \mathbf{y} , à la distance δ . Le point d'apprentissage suivant, \mathbf{z} , n'est pas à distance inférieure à δ de \mathbf{x} . Aucun point du type \mathbf{v}_1 ou \mathbf{v}_2 ne peut plus être le ppv de \mathbf{x} . En revanche, il faudra calculer la distance $\Delta(\mathbf{x}, \mathbf{v}_3)$.

tissage. Si $\Delta(\mathbf{x}, \mathbf{z}) \leq \delta$, on réactualise δ et \mathbf{y} . Sinon, on peut affirmer que parmi tous les points d'apprentissage restant à examiner, on doit définitivement éliminer deux catégories :

- ceux qui sont situés à l'intérieur de la boule de centre \mathbf{z} et de rayon $\Delta(\mathbf{x}, \mathbf{z}) - \delta$.
- ceux qui sont à l'extérieur de la boule de centre \mathbf{z} et de rayon $\Delta(\mathbf{x}, \mathbf{z}) + \delta$.

En effet, pour tout point \mathbf{v} , le triangle $(\mathbf{x}, \mathbf{v}, \mathbf{z})$ vérifie :

$$\Delta(\mathbf{x}, \mathbf{z}) \leq \Delta(\mathbf{x}, \mathbf{v}) + \Delta(\mathbf{v}, \mathbf{z})$$

Si \mathbf{v} est un point d'apprentissage restant appartenant à la première catégorie, on a :

$$\Delta(\mathbf{v}, \mathbf{z}) \leq \Delta(\mathbf{x}, \mathbf{z}) - \delta$$

soit :

$$\Delta(\mathbf{v}, \mathbf{z}) + \delta \leq \Delta(\mathbf{x}, \mathbf{z})$$

Donc en combinant les deux inégalités :

$$\Delta(\mathbf{v}, \mathbf{z}) + \delta \leq \Delta(\mathbf{x}, \mathbf{v}) + \Delta(\mathbf{v}, \mathbf{z})$$

d'où :

$$\delta \leq \Delta(\mathbf{x}, \mathbf{v})$$

ce qui prouve que le point \mathbf{v} ne peut pas être le plus proche voisin de \mathbf{x} .

Un raisonnement analogue mène à éliminer les points appartenant à la seconde catégorie (voir la figure 14.18).

Algorithme 14.4 Recherche rapide du plus proche voisin

Début

On cherche le ppv de \mathbf{x} dans $A = \{\mathbf{y}_1, \dots, \mathbf{y}_m\}$

Calculer toutes les distances $\Delta(\mathbf{y}_i, \mathbf{y}_j)$; $\delta = +\infty$; $i = 1$;

tant que $i < m$ et \mathbf{y}_i non marqué faire

 Calculer $\Delta(\mathbf{x}, \mathbf{y}_i)$

si $\Delta(\mathbf{x}, \mathbf{y}_i) < \delta$ **alors**

$\delta = \Delta(\mathbf{x}, \mathbf{y}_i)$; $ppv = \mathbf{y}_i$;

sinon

$j = i + 1$;

tant que $j \leq m$ **faire**

si $\Delta(\mathbf{y}_i, \mathbf{y}_j) \leq \Delta(\mathbf{x}, \mathbf{y}_i) - \delta$ **ou** $\Delta(\mathbf{y}_i, \mathbf{y}_j) \geq \Delta(\mathbf{x}, \mathbf{y}_i) + \delta$ **alors**

 marquer \mathbf{y}_j

fin si

fin tant que

fin si

fin tant que

Fin

En utilisant ce principe brutalement, il faut pour commencer calculer $m(m - 1)/2$ distances, ce qui prend évidemment du temps ; cependant, ce calcul n'est à faire qu'une seule fois, alors que la décision de classement d'un point inconnu est accélérée systématiquement.

On peut grandement améliorer cette technique, soit en sélectionnant un sous-ensemble de points d'apprentissage, soit en combinant le principe avec celui d'une organisation arborescente de l'ensemble d'apprentissage pour appliquer une technique de séparation-évaluation *branch and bound*. Une revue de ces méthodes est donnée dans [BB92]. On trouve en particulier dans [Vid94b] une méthode sophistiquée (LAESA) dont la complexité est en pratique indépendante de m .

Un exemple

Plaçons nous dans \mathbb{R}^2 , avec l'ensemble d'apprentissage suivant (la classe des exemples n'a pas besoin d'être précisée ici) :

$$\mathcal{S} = \left\{ \mathbf{y}_1 = \begin{pmatrix} 5 \\ 4 \end{pmatrix}, \mathbf{y}_2 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \mathbf{y}_3 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \mathbf{y}_4 = \begin{pmatrix} 0 \\ 8 \end{pmatrix}, \mathbf{y}_5 = \begin{pmatrix} 3 \\ 4 \end{pmatrix} \right\}$$

On calcule d'abord la demi-matrice des distances entre les exemples. Ici, on a tabulé le carré de ces distances :

	y_1	y_2	y_3	y_4	y_5
y_1	0	41	34	41	4
y_2		0	1	64	25
y_3			0	49	18
y_4				0	25
y_5					0

Soit le point $\mathbf{x} = \begin{pmatrix} 3 \\ 3 \end{pmatrix}$ dont on cherche le plus proche voisin dans A .

On calcule $\Delta(\mathbf{x}, y_1) = \sqrt{2}$, et y_1 devient le plus proche voisin courant, à la distance $\delta = \sqrt{2}$.

Le lecteur est invité ici à tracer le cercle de centre \mathbf{x} et de rayon δ ainsi que les deux cercles de centre y_2 , tangents intérieurement et extérieurement au premier cercle, puis à placer les quatre autres points de \mathcal{S} sur cette figure.

On calcule ensuite $\Delta(\mathbf{x}, y_2) = \sqrt{18}$ qui est strictement supérieur à δ . y_2 n'est donc pas le nouveau ppv. Quels points de \mathcal{S} peut-on d'ores et déjà éliminer (*marquer*) ?

y_3 ? Oui, car $\Delta(y_3, y_2) \leq \Delta(\mathbf{x}, y_2) - \delta$ (puisque $\sqrt{1} \leq \sqrt{18} - \sqrt{2}$).

y_4 ? Oui, car $\Delta(y_4, y_2) \geq \Delta(\mathbf{x}, y_2) + \delta$ (puisque $\sqrt{64} \geq \sqrt{18} + \sqrt{2}$).

y_5 ? Non⁹, car on n'a :

- ni $\Delta(y_5, y_2) \leq \Delta(\mathbf{x}, y_2) - \delta$ (puisque $\sqrt{25} > \sqrt{18} - \sqrt{2}$).
- ni $\Delta(y_5, y_2) \geq \Delta(\mathbf{x}, y_2) + \delta$ (puisque $\sqrt{25} < \sqrt{18} + \sqrt{2}$).

On passe au point d'apprentissage suivant non marqué, y_5 et on constate que $\Delta(\mathbf{x}, y_5) = 1 < \delta$. y_5 est donc le ppv de \mathbf{x} dans \mathcal{S} .

14.4 Les méthodes semi paramétriques

Nous supposons dans cette section que, contrairement aux méthodes d'estimation paramétriques, nous ne connaissons pas *a priori* la forme analytique des distributions de probabilités. En revanche, nous supposons cependant que ces distributions suivent des lois dont les « hyper paramètres » peuvent être déterminés de manière systématique.

Il n'existe pas à notre connaissance de catalogue exhaustif des méthodes semi paramétriques. Il risquerait fort de toute façon d'être rapidement obsolète. Nous avons choisi ici de présenter succinctement trois méthodes pour leur intérêt dans les applications pratiques.

14.4.1 La discrimination logistique

La discrimination logistique s'intéresse aux problèmes à deux classes. Elle est issue de l'observation suivante : pour de nombreuses formes de distributions, la règle de décision bayésienne peut se ramener à une équation linéaire du type :

$$\ln \frac{p(\mathbf{x}|\omega_1)}{p(\mathbf{x}|\omega_2)} = \boldsymbol{\theta}^\top \mathbf{x} + \theta_0 \quad (14.29)$$

où $\boldsymbol{\theta}$ est un vecteur de p paramètres et θ_0 une constante.

9. À noter que s'il existait dans l'ensemble d'apprentissage le point $y_6 = \begin{pmatrix} -5 \\ -4 \end{pmatrix}$, il ne serait pas éliminé non plus, bien que très éloigné de \mathbf{x} .

Nous en avons vu un exemple dans la section 14.2.2 pour des distributions normales.

Cette approche ne suppose pas de forme analytique pour les distributions d'appartenance aux classes, mais seulement pour leur rapport $p(\mathbf{x}|\omega_1)/p(\mathbf{x}|\omega_2)$, ce qui est une hypothèse beaucoup moins forte. Comme un grand nombre de distributions statistiques courantes vérifient cette condition, elle est raisonnable et intéressante.

Par exemple, toutes les lois de probabilité de la famille exponentielle:

$$p(\mathbf{x}|\boldsymbol{\theta}_k) = \exp \boldsymbol{\theta}_k^\top \mathbf{x} + a(\mathbf{x}) + b_k(\boldsymbol{\theta}_k)$$

respectent cette hypothèse. Ceci inclut aussi bien des distributions de lois continues que des distributions de lois discrètes, par exemple les lois normales (avec égalité des matrices de covariance), la loi gamma, la loi bêta, la loi de Poisson, la loi binomiale, la loi multinomiale, etc.

Dans tous ces cas, la fonction de décision prend la forme de l'inéquation :

$$d(\mathbf{x}) = \ln \frac{p(\mathbf{x}|\omega_1)}{p(\mathbf{x}|\omega_2)} + \ln \frac{(l_{21} - l_{11}) p(\omega_1)}{(l_{12} - l_{22}) p(\omega_2)} \geq 0$$

où l'on attribue la forme \mathbf{x} à la classe ω_1 si $d(\mathbf{x}) \geq 0$ et à la classe ω_2 sinon. Soit, par rapprochement avec l'équation (14.29) :

$$\forall \mathbf{x}, \quad \boldsymbol{\theta}^\top \mathbf{x} + \theta_0 - \ln \frac{(l_{21} - l_{11}) p(\omega_1)}{(l_{12} - l_{22}) p(\omega_2)} \begin{cases} \geq 0 & \text{alors } \mathbf{x} \text{ est attribuée à } \omega_1, \\ < 0 & \text{alors } \mathbf{x} \text{ est attribuée à } \omega_2 \end{cases} \quad (14.30)$$

Cette règle de décision fait clairement ressortir que la fonction de décision est linéaire en \mathbf{x} . De plus, c'est la fonction de décision qui est paramétrée, et non, comme dans les approches paramétrées, les vraisemblances conditionnelles $p(\mathbf{x}|\omega_i)$.

Par ailleurs, il est aisément de montrer que l'équation (14.29) est équivalente à une comparaison de fonctions de type sigmoïde :

$$\begin{aligned} p(\omega_1|\mathbf{x}) &= \frac{e^{d(\mathbf{x})}}{1 + e^{d(\mathbf{x})}} \\ p(\omega_2|\mathbf{x}) &= \frac{1}{1 + e^{d(\mathbf{x})}} \end{aligned} \quad (14.31)$$

Nous avons déjà rencontré des expressions de ce type appelées fonctions logistiques, dans le cadre des réseaux connexionnistes (chapitre 10). Dans ce cadre, elles sont appelées *fonctions logistiques*. On voit d'après ces deux inégalités que si l'approche bayésienne vue plus haut s'appuie sur les distributions conditionnelles $p(\mathbf{x}|\omega_i)$, l'approche de la discrimination logistique s'appuie en revanche sur les probabilités des classes $P(\omega_i|\mathbf{x})$.

Pour calculer les paramètres θ_0 et $\boldsymbol{\theta}$, il faut utiliser un échantillon d'apprentissage \mathcal{S} constitué d'exemples de la classe ω_1 et d'exemples de la classe ω_2 . Anderson, dans [And82], a montré que ces paramètres peuvent être obtenus par maximisation de la vraisemblance des paramètres conditionnellement aux exemples :

$$\begin{aligned} L &= \prod_{\mathbf{x} \in \omega_1} p(\omega_1|\mathbf{x}) \prod_{\mathbf{x} \in \omega_2} p(\omega_2|\mathbf{x}) \\ &= \prod_{\mathbf{x} \in \omega_1} \frac{e^{d(\mathbf{x})}}{1 + e^{d(\mathbf{x})}} \prod_{\mathbf{x} \in \omega_2} \frac{1}{1 + e^{d(\mathbf{x})}} \end{aligned}$$

Il a été démontré que sous des conditions très générales (voir [Alb78]), le maximum de L est unique.

Cette approche se généralise facilement au cas multi classes. Il faut alors considérer les fonctions :

$$\log \frac{p(\mathbf{x}|\omega_i)}{p(\mathbf{x}|\omega_j)} = \boldsymbol{\theta}_i^\top \mathbf{x} + \theta_{0i}$$

14.4.2 Les mélanges de distributions

Dans la section précédente, nous avons vu qu'une approche pour rendre plus souple le type de distributions qu'il est possible d'approximer tout en conservant un moyen de contrôle sur l'espace d'hypothèses est d'utiliser des fonctions de discrimination paramétrables, comme la fonction logistique. Une autre approche consiste à supposer que les distributions peuvent être décomposées en un produit de distributions plus simples. Dans les modèles de mélanges de distributions, une distribution complexe p est paramétrée comme une combinaison linéaire de distributions plus simples, souvent la distribution normale, sous la forme :

$$p(\mathbf{x}) = \sum_{i=1}^M \lambda_i p_i(\mathbf{x}) \quad (14.32)$$

où les $\lambda_i \geq 0$ sont appelés coefficients de mélange et satisfont à la condition : $\sum_i \lambda_i = 1$. Les distributions p_i sont appelées les composantes de mélange et ont leurs propres paramètres (moyenne, écart type, etc.). Remarquons le lien avec les fonctions noyau : on peut avoir $p_i = K(\mathbf{x} - \mathbf{x}_i)$ où \mathbf{x}_i est le centre du noyau et $M = m$. Il y a dans ce cas autant de composantes au mélange que de points d'apprentissage.

Quelques points sont à noter :

- Les mélanges de distributions ne prennent pas en compte directement l'étiquette des exemples. Ce sont des moyens d'exprimer des densités de probabilités. Leur estimation ressort donc des techniques d'apprentissage non supervisé (voir au chapitre 15). Il est cependant possible de les utiliser pour des tâches de classification en estimant la distribution de probabilités pour chaque classe tour à tour :

$$p(\mathbf{x}|\omega_k) = \sum_{i=1}^M \lambda_i p_i(\mathbf{x}|\omega_k) \quad (14.33)$$

- Une propriété importante des mélanges de distributions est que pour un large choix de fonctions de base, elles permettent d'approximer avec un degré arbitraire de précision n'importe quelle distribution continue, du moment que le mélange a un nombre suffisant de composantes et que les paramètres sont bien choisis [MB88a].
- Un choix usuel pour les distributions composantes ou fonctions de base est de prendre des fonctions gaussiennes représentant la probabilité conditionnelle d'observer \mathbf{x} quand la classe est ω_k : $p(\mathbf{x}|\omega_k)$. La plupart des ouvrages comportant une section sur ces méthodes traitent de ce cas (par exemple [Bis95]).
- L'idée des approches semi paramétriques est de faire varier systématiquement le nombre de paramètres du modèle en fonction de la difficulté du problème traité. Dans le cas des mélanges de distributions, ce principe se traduit par une procédure de choix du nombre M de fonctions de base utilisées dans le mélange. Malheureusement, il semble que ce choix soit un problème notablement difficile [MB88a, FL94].

- Concernant l'apprentissage d'un mélange de distributions, il est possible d'interpréter les coefficients de mélange λ_i comme des probabilités *a priori* des composantes du mélange. Dans ce cas, pour un point \mathbf{x}_l donné, il est possible d'utiliser le théorème de Bayes pour évaluer la probabilité *a posteriori* correspondante :

$$R_{li} \equiv p(i|\mathbf{x}_l) = \frac{\lambda_i p(\mathbf{x}_l|i)}{\sum_j \lambda_j p(\mathbf{x}_l|j)} \quad (14.34)$$

La valeur de $p(i|\mathbf{x}_l)$ peut être vue comme la responsabilité que la composante i assume pour « expliquer » la donnée \mathbf{x}_l (d'où la notation R_{li}). En prenant le logarithme de cette expression on obtient la vraisemblance :

$$\mathcal{L}(\{\pi_i, p_i\}) = \sum_{l=1}^m \log \left\{ \sum_{i=1}^M \pi_i p(\mathbf{x}_l|i) \right\} \quad (14.35)$$

Malheureusement, la maximisation de cette vraisemblance est beaucoup plus complexe que pour une distribution à une seule composante à cause de la somme dans le logarithme. Un algorithme élégant et puissant pour réaliser cette optimisation est l'algorithme *EM* (voir l'annexe 18.9).

14.4.3 Le cas des réseaux connexionnistes et des arbres de décision

Les réseaux connexionnistes et les arbres de décision sont aussi des familles d'estimateurs semi paramétriques. Rappelons que les premiers peuvent servir de mécanisme (sophistiqué) pour estimer une densité de probabilité (voir le chapitre 10, paragraphe 10.3.5).

Quant aux seconds, on peut considérer qu'ils réalisent une estimation à valeur constante dans chaque feuille de l'arbre. Un arbre de décision, en effet, partitionne l'espace \mathcal{X} des observations possibles récursivement en sous-régions jusqu'à des sous-régions suffisamment homogènes correspondant aux feuilles de l'arbre et dont l'étiquette sert à étiqueter les formes qui s'y trouvent.

Nous n'approfondissons pas ces aspects ici mais nous conseillons de consulter la référence [CM98].

Notes historiques et sources bibliographiques

Nous nous sommes contentés ici d'un survol. Plusieurs publications sont à recommander pour aller plus loin : [EH81, TSM85, RW84].

E. Parzen a proposé en 1962 ([Par62]) d'estimer les densités de probabilités par des superpositions de « fenêtres ». Un grand nombre de travaux de probabilistes et de statisticiens ont approfondi cette approche. Les applications en apprentissage ont connu un renouveau avec l'application à certains types de réseaux connexionnistes ([Web99]), fondés sur des fonctions dites « radiales de base » qui sont des fenêtres de Parzen de type particulier.

Les méthodes des k-plus proches voisins pour la reconnaissance des formes remontent aux années 1950, mais les algorithmes et les preuves de convergence sont un peu plus récentes : T. Cover et P. Hart en ont posé les principes en 1967 ([CH67]). Le livre de J. Kittler et P. Devijver ([DK82]) a apporté un grand nombre de résultats théoriques et pratiques supplémentaires. Les algorithmes rapides sont nombreux, comme on peut le voir en lisant la compilation de B. Dasarathy ([Das90]). L'algorithme AESA et ses variantes, les meilleurs du genre, proviennent de E. Vidal et de son équipe ([Vid94b]). On lira avec intérêt les articles recueillis par D. Aha ([Aha97]) où la technique « paresseuse » de la recherche de formes similaires dans un ensemble d'exemples est étendue à de nombreux domaines de l'apprentissage, y compris à des données symboliques.

Résumé

- L'apprentissage bayésien consiste à partir d'hypothèses *a priori* pour les réviser en fonction des données d'apprentissage.
- Cette opération est optimale au sens probabiliste: les hypothèses *a posteriori* obtenues de la sorte sont en effet les plus vraisemblables.
- L'apprentissage bayésien requiert d'une part une connaissance *a priori* sur la vraisemblance des hypothèses en concurrence et d'autre part celle la probabilité des données d'apprentissage conditionnellement à ces hypothèses. Ces valeurs doivent être estimées à partir de l'ensemble d'apprentissage.
- Les méthodes d'estimation paramétrique font l'hypothèse que la distribution à estimer possède une certaine forme analytique et trouvent les meilleurs paramètres.
- Les méthodes d'estimation non-paramétrique estiment une densité conditionnelle en un point en examinant comment l'ensemble d'apprentissage se comporte au voisinage de ce point.
- Les méthodes des k -plus proches voisins ont l'avantage de la simplicité. Une algorithme efficace existe pour les rendre rapides.

Chapitre 15

La classification non supervisée et la découverte automatique

Jusqu'ici, il n'a été question dans cet ouvrage que d'apprentissage supervisé : les exemples ont toujours été pourvus d'une étiquette ou d'une valeur numérique fournie par un oracle (un expert). Dans ce chapitre, nous nous plaçons en dehors de cette hypothèse, afin d'aborder deux problèmes particuliers d'apprentissage : la classification non supervisée et la découverte automatique.

La problématique de la classification automatique est simple : étant donné un certain nombre d'objets décrits par des attributs, est-il possible d'identifier les familles dans lesquelles se regroupent naturellement ces objets ? Techniquement, il y a deux manières de l'aborder : soit organiser les données selon une hiérarchie de classes ou de familles, comme font par exemple les naturalistes ; soit faire l'hypothèse qu'il existe un certain nombre de classes dans les données et chercher à les partitionner le mieux possible en autant de sous-ensembles disjoints.

La découverte automatique peut se comprendre comme une technique particulière de régression. Il s'agit de trouver les lois les plus simples possible pour expliquer des phénomènes naturels ou des invariants dans les bases de données. Idéalement, un programme de découverte automatique pourrait par exemple retrouver la loi d'Ohm à partir de mesures sur l'intensité du courant, la valeur de la résistance électrique et la différence de potentiel. Mieux, il pourrait établir des lois inconnues ou mal connues en fouillant des grosses bases de données, par exemple une relation algébrique entre l'âge d'un client, sa situation géographique et la marque des pneus de sa voiture.

Les méthodes de classification automatique et de découverte automatique, donc l'apprentissage non supervisé, constituent par conséquent la base des techniques de la fouille de données. Le dernier paragraphe de ce chapitre s'intéresse directement à la recherche d'associations entre les attributs binaires dans une base de données. Ces associations peuvent être ou non associées à la date d'enregistrement de l'exemple dans la base. C'est désormais une des techniques les plus fécondes en fouille de données.

LES ANIMAUX se divisent en a) appartenant à l'empereur, b) embaumés, c) apprivoisés, d) cochons de lait, e) sirènes, f) fabuleux, g) chiens en liberté, h) inclus dans la présente classification, i) qui s'agitent comme des fous, j) innombrables, k) dessinés avec un pinceau très fin en poil de chameau, l) *et cætera*, m) qui viennent de casser la cruche, n) qui de loin semblent des mouches.

Cette classification que M. Foucault qualifie de « déconcertante »¹ est due à l'imagination de J.-L. Borges². Son étrangeté provient d'abord de ce qu'il ne s'agit pas vraiment d'une « division » (en terme formel : d'une partition) des espèces : un animal peut être à la fois un cochon de lait, apprivoisé et une possession de l'empereur. Surtout, peut-être, la variété des concepts utilisés pour regrouper les espèces est absurde : « s'agiter » ne s'oppose ou ne se compare en rien à « être dessiné », par exemple. Et que dire de « inclus dans la présente classification » ou de « *et cætera* » ? Par contraste, une classification raisonnable des animaux devrait utiliser une description comparable de chaque animal (vertébré ou non, et si vertébré, mammifère ou non, etc.) et *in fine* fournir une partition opératoire (par exemple, deux espèces sont différentes si elles ne sont pas interfécondes, pour des raisons génétiques ou géographiques).

Nous avons parlé jusqu'ici dans ce livre de classification supervisée, pour laquelle les éléments de l'ensemble d'apprentissage sont pourvus d'une étiquette ou d'une valeur numérique fournies par un oracle. Dans ce chapitre, la problématique de la classification est différente : étant donné un certain nombre d'objets décrits par des attributs, est-il possible d'identifier les familles dans lesquelles se regroupent ces objets ? C'est pour ainsi dire le problème auquel l'oracle a été confronté avant d'être capable de réaliser un étiquetage. Ce problème s'appelle la classification non supervisée ou le *clustering*³. Il y a deux manières de l'aborder : soit en construisant une hiérarchie de classes ; soit directement en cherchant un nombre fixé de classes.

Ces deux méthodes sont assez aisées à pratiquer pour des attributs numériques, quand la notion de distance est naturelle. Dans le cas d'attributs binaires ou nominaux, un certain nombre d'extensions sont cependant possibles.

Pour finir, ce chapitre abordera un autre problème d'apprentissage non supervisé : celui de la découverte automatique de lois à partir de données, une forme empirique et constructive de régression non linéaire. Ce domaine est étudié depuis longtemps et gagne en importance actuellement grâce à l'apparition de la fouille de données.

Nous traitons ici l'apprentissage supervisé de manière rapide. Le lecteur désirant approfondir ses connaissances dans ce domaine peut se reporter aux références proposées dans la section 15.6.2.

1. « On sait ce qu'il y a de déconcertant dans la proximité des extrêmes ou tout bonnement le voisinage soudain des choses sans rapport ; l'énumération qui les entrechoque possède à elle seule un pouvoir d'enchantement. » *Les mots et les choses*. Gallimard (1966)

2. *La langue analytique de John Wilkins*, dans *Enquêtes*, Gallimard (1957).

3. On dit parfois « la coalescence » ou simplement « la classification ».

15.1 La classification hiérarchique de données numériques

15.1.1 Généralités

Soit un ensemble $\mathcal{S} = \{\mathbf{x}_1 \dots \mathbf{x}_m\}$ de m objets. Nous allons définir formellement une hiérarchie sur \mathcal{S} , de deux manières différentes, mais finalement équivalentes. Rappelons d'abord la définition d'une partition et celle de la relation d'ordre associée. Les deux termes *sous-ensemble* et *partie* sont ici synonymes.

Définition 15.1 (Partition)

Soit un ensemble fini \mathcal{S} . Une partition π de \mathcal{S} est un ensemble de parties de \mathcal{S} , non vides et disjointes deux à deux, dont l'union est \mathcal{S} . Si s désigne un élément de \mathcal{S} , il existe donc un unique élément, ou bloc, de π comprenant s . Une partition π_i est plus fine qu'une partition π_j si et seulement si tout bloc de π_j est un bloc de π_i ou est l'union de plusieurs blocs de π_i .

Selon la relation d'ordre précédente, la partition la plus fine de \mathcal{S} est constituée de m blocs et s'écrit :

$$\mathcal{P}_m(\mathcal{S}) = (\mathbf{x}_1), \dots, (\mathbf{x}_m)$$

alors que la partition la moins fine n'a qu'un bloc et s'écrit :

$$\mathcal{P}_1(\mathcal{S}) = (\mathbf{x}_1, \dots, \mathbf{x}_m)$$

Par exemple, pour $\mathcal{S} = \{a, b, c, d, e, f\}$, les deux partitions

$$(a, b, c), (d), (e, f) \text{ et } (a, b), (c, d), (e, f)$$

sont toutes deux plus fines que la partition

$$(a, b, c, d), (e, f)$$

mais n'ont pas de relation de finesse entre elles.

Définition 15.2

Une chaîne dans l'ensemble des partitions de \mathcal{S} est un ensemble de partitions $\{\pi_1, \dots, \pi_r\}$ tel que pour $i = 1, r - 1$ on a : π_i est plus fine que π_{i+1} .

Définition 15.3 (Hiérarchie, 1)

Une hiérarchie sur \mathcal{S} est une chaîne de partitions de \mathcal{S} dont la plus fine est \mathcal{P}_m et la moins fine est \mathcal{P}_1 .

L'autre définition est la suivante :

Définition 15.4 (Hiérarchie, 2)

Une hiérarchie H sur \mathcal{S} est un sous-ensemble des parties de \mathcal{S} tel que :

- pour tout élément \mathbf{x} de \mathcal{S} , $\{\mathbf{x}\} \in H$;
- pour tout couple d'éléments h et h' de H avec $h \neq h'$, on a :
 - soit $h \cap h' = \emptyset$,
 - soit $h \cap h' \neq \emptyset$, alors soit $h \subset h'$, soit $h' \subset h$.

Par exemple, la hiérarchie représentée à la figure 15.1 peut être vue soit comme la chaîne de partitions

$$\begin{aligned} & (a, b, c, d, e, f) \\ & (a, b, c, d), (e, f) \\ & (a, b, c), (d), (e, f) \\ & (a, b, c), (d), (e), (f) \\ & (a), (b), (c), (d), (e), (f) \end{aligned}$$

soit comme l'ensemble de parties de \mathcal{S} : $H = \{h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9, h_{10}\}$, avec

$$\begin{array}{ll} h_1 = \{a\} & h_7 = \mathcal{S} = \{a, b, c, d, e, f\} \\ h_2 = \{b\} & h_8 = \{a, b, c, d\} \\ h_3 = \{c\} & h_9 = \{e, f\} \\ h_4 = \{d\} & h_{10} = \{a, b, c\} \\ h_5 = \{e\} & \\ h_6 = \{f\} & \end{array}$$

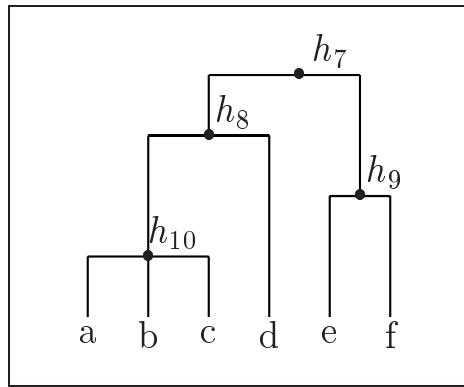


FIG. 15.1 – Une hiérarchie sur $\mathcal{S} = \{a, b, c, d, e, f\}$.

Définition 15.5 (Hiérarchie indicée monotone)

Une hiérarchie indicée est une hiérarchie H sur un ensemble fini à laquelle on associe une suite de nombres réels r_i . Une hiérarchie indicée est monotone si pour deux éléments h_i et h_{i+1} consécutifs dans H , avec h_i plus fine que h_{i+1} , on a $r_i \leq r_{i+1}$.

Par exemple, sur la figure 15.2, la hiérarchie de la figure 15.1 a été indicée de manière monotone sur l'axe vertical selon l'association :

$$\begin{array}{ll} (a, b, c, d, e, f) & 4.2 \\ (a, b, c, d) & 3 \\ (e, f) & 1.9 \\ (a, b, c) & 1.1 \\ (a), (b), (c), (d), (e), (f) & 0 \end{array}$$

Construire une hiérarchie sur un ensemble d'exemples est donc équivalent à trouver une chaîne de partitions sur cet ensemble. Si l'on construit une hiérarchie indicée monotone sur cet ensemble, on peut obtenir les partitions de la chaîne en « coupant » la hiérarchie pour une valeur de l'indice. Par exemple, sur la figure 15.2, la coupure pour la valeur 2.5 de l'indice fournit la partition $(a, b, c), (d), (e, f)$.

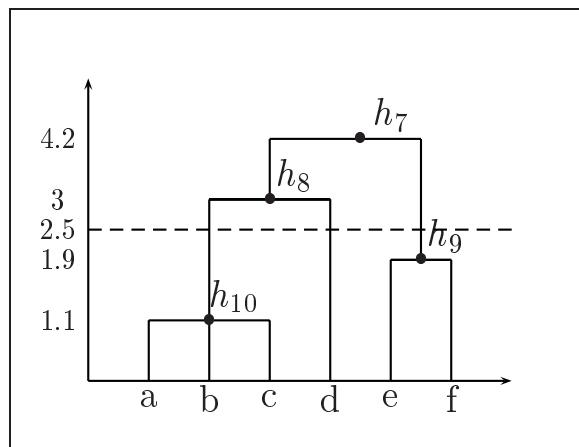


FIG. 15.2 – Une hiérarchie indicée sur $\mathcal{S} = \{a, b, c, d, e, f\}$. La coupure pour la valeur 2.5 de l'indice produit la partition $(a, b, c), (d), (e, f)$.

On peut associer à toute hiérarchie indicée H une mesure de dissimilarité entre ses éléments de la façon suivante: la dissimilarité $\delta(h_i, h_j)$ entre les parties h_i et h_j de H prend la valeur de l'indice de la plus petite partie h de H qui contient h_i et h_j .

On peut démontrer que cette mesure de dissimilarité possède une propriété forte: elle est une *distance ultramétrique*.

Définition 15.6 (Ultramétrique)

Une ultramétrique (*abrégé de distance ultramétrique*) sur un ensemble Z est une application de $Z \times Z \rightarrow \mathbb{R}$ qui vérifie les trois conditions suivantes pour tous les z_i, z_j et z_k de Z :

- $\delta(z_i, z_i) = 0$;
- $\delta(z_i, z_j) = \delta(z_j, z_i)$;
- $\delta(z_i, z_k) \leq \text{Max}(\delta(z_i, z_j), \delta(z_j, z_k))$.

Par exemple, la hiérarchie indicée de la figure 15.2 place les éléments $\{a\}$ et $\{d\}$ à la distance ultramétrique $\delta(\{a\}, \{d\}) = 3$, puisque la plus petite partie de H qui contient $\{a\}$ et $\{d\}$ est h_8 . De même, $\delta(\{d\}, \{e, f\}) = 4.2$ puisque la plus petite partie de H qui contient $\{d\}$ et $\{e, f\}$ est h_7 . On peut vérifier la propriété ultramétrique:

$$\delta(\{a\}, \{e, f\}) \leq \text{Max}(\delta(\{a\}, \{d\}), \delta(\{d\}, \{e, f\})) = \text{Max}(3, 4.2) = 4.2$$

En résumé, nous sommes désormais en possession des notions suivantes :

- Nous avons un ensemble d'exemples \mathcal{S} composé de m objets.
- Cet ensemble est muni d'une distance Δ , en général la distance euclidienne.
- Nous cherchons à construire une hiérarchie indicée H de partitions de \mathcal{S} .
- Nous savons que construire une telle hiérarchie indicée est équivalent à donner une ultramétrique δ sur les parties de \mathcal{S} présentes dans H .
- Si la hiérarchie indicée construite est monotone, la hiérarchie de partitions trouvée est complètement satisfaisante, puisque l'agrégation de deux sous-ensembles d'une partition produit une partition d'indice supérieur.

Il nous faut donc utiliser la distance Δ sur \mathcal{S} pour en déduire une hiérarchie indicée, si possible monotone. Pour cela, il faut définir une autre mesure D de dissimilarité, celle-là entre tous les

couples de parties de \mathcal{S} . La première étape est facile: si l'on considère les objets de \mathcal{S} comme des parties de \mathcal{S} comportant un seul élément, on posera naturellement :

$$D(\{\mathbf{x}\}, \{\mathbf{y}\}) = \Delta(\mathbf{x}, \mathbf{y})$$

Il nous faut ensuite trouver comment définir la valeur de D sur n'importe quel couple de sous-ensembles de \mathcal{S} . Nous présentons dans les paragraphes suivants quelques indices D qui permettent de construire une hiérarchie sur \mathcal{S} . Auparavant, donnons un algorithme constructif qui utilise D et dont le résultat est une hiérarchie indicée, mais pas forcément monotone.

15.1.2 Un algorithme général de classification hiérarchique

Quand on dispose d'une mesure de similarité D entre les parties de \mathcal{S} , la construction d'une hiérarchie indicée se fait de manière simple par l'algorithme 15.1.

Chaque étape de cet algorithme produit un élément de la chaîne de partitions. Le nombre maximal d'étapes est m , le nombre d'objets.

Il nous reste maintenant à présenter quelques mesures de dissimilarité ou *indices D* classiques qui produisent par cet algorithme des hiérarchies indicées.

Algorithme 15.1 Algorithme de classification hiérarchique

Etablir la table T_D des valeurs de $D(\mathbf{x}, \mathbf{y})$ pour \mathbf{x} et \mathbf{y} parcourant \mathcal{S} .

tant que la table T_D a plus d'une colonne faire

Choisir les deux sous-ensembles h_i, h_j de \mathcal{S} tels que $D(h_i, h_j)$ est le plus petit nombre réel dans la table T_D

Supprimer h_j de la table, remplacer h_i par $h_i \cup h_j$

Calculer les mesures de similarité D entre $h_i \cup h_j$ et les autres éléments de la table.

fin tant que

15.1.3 L'indice du lien simple

Cet indice produit l'ultramétrique dite *sous-dominante*. Il mesure la distance euclidienne entre les deux points les plus proches, l'un dans un bloc de partition, l'autre dans le second. Quand les blocs sont réduits à un élément, cet indice mesure la distance euclidienne entre ces deux éléments. La hiérarchie indicée trouvée est monotone.

Reprenons un ensemble à six exemples et donnons la table des distances euclidiennes entre les exemples. Cette dernière est aussi la table T_D , qui est identique sur les couples d'objets. Comme cette table est symétrique, seule une moitié est représentée.

	a	b	c	d	e	f
a	0	1.1	1.1	3	6	5
b		0	1.1	4	5.5	4.2
c			0	3	6.5	5.3
d				0	9	8
e					0	1.9
f						0

L'algorithme de classification hiérarchique produit successivement les tables suivantes (la plus petite valeur non nulle de chaque table est en gras) :

D	$h_{10} = \{a, b\}$	$\{c\}$	$\{d\}$	$\{e\}$	$\{f\}$
h_4	0	1.1	3	5.5	4.2
c		0	3	6.5	5.3
d			0	9	8
e				0	1.9
f					0

D	$h_{10} = \{a, b, c\}$	$\{d\}$	$\{e\}$	$\{f\}$
h_4	0	3	5.5	4.2
d		0	6.5	5.3
e			0	1.9
f				0

D	h_{10}	$\{d\}$	$h_9 = \{e, f\}$
h_4	0	3	4.2
d		0	5.3
h_3			0

D	$h_8 = h_{10} \cup \{d\}$	h_9
h_2	0	4.2
h_3		0

D	$h_7 = h_{10} \cup h_9$
h_1	0

On retrouve donc la hiérarchie indiquée des figures 15.1 et 15.2.

Cette indice est très simple et semble naturel. Il peut donner des résultats surprenants en raison d'un certain « effet de chaîne », qui regroupe parfois des points de manière non naturelle. On utilise de préférence pour cette raison des indices un peu plus sophistiqués, comme ceux qui sont proposés aux paragraphes suivants.

15.1.4 L'indice de la distance entre centres de gravité

Prendre pour indice D la distance euclidienne entre les centres de gravité des blocs fournit une hiérarchie non forcément monotone, ce qui n'est en général pas souhaitable.

Il est facile de la remarquer sur l'exemple suivant : soient les points $\{a, b, c\}$ de \mathbb{R}^2 de coordonnées $a = (0, 0)$, $b = (9, 0)$ et $c = (4.5, 8.5)$. La distance la plus petite est entre a et b et vaut 9 (les deux autres valent environ 9.6). Le centre de gravité du bloc (a, b) , de coordonnées $(4.5, 0)$ est à une distance de c qui vaut 8.5, strictement à 9.

Par conséquent, il se produit ce qu'on appelle une *inversion* dans la construction de la hiérarchie : le résultat n'est pas une hiérarchie indiquée monotone.

15.1.5 L'indice de Ward

Pour tenir compte de la variance des classes et pour éviter l'effet de chaîne dans la classification, Ward [Sim85] a proposé d'utiliser l'indice suivant, donné ici sous la forme de « formules de réactualisation ». Il fournit une hiérarchie indiquée monotone.

Nous sommes à l'étape courante de l'algorithme 15.1, nous avons choisi les deux blocs les plus proches h_i et h_j et nous cherchons à calculer l'indice entre le nouveau bloc $h_i \cup h_j$ et un autre bloc h_k . Le nombre d'éléments du bloc h_i (respectivement : h_j , h_k) vaut m_i (respectivement : m_j , m_k).

$$D(h_i \cup h_j, h_k) = \frac{n_k + n_i}{n_k + n_i + n_j} D(h_i, h_k) + \frac{n_k + n_j}{n_k + n_i + n_j} D(h_j, h_k) - \frac{n_i + n_j}{n_k + n_i + n_j} D(h_i, h_j)$$

15.1.6 L'indice de la vraisemblance du lien

Lerman [Ler81] propose de mesurer l'indice entre deux blocs h_i et h_j par la formule :

$$D\epsilon(h_i, h_j) = -\text{Log}(-\text{Log}([s(h_i, h_j)]^{n_i n_j \epsilon}))$$

où $s(h_i, h_j)$ est l'indice du lien simple entre h_i et h_j , c'est-à-dire la distance⁴ entre les deux objets les plus proches, l'un dans h_1 , l'autre dans h_2 .

Ce calcul a pour effet, de manière contrôlable par ϵ , de faciliter le regroupement des classes à variance faible. Comme l'indice de Ward, il est fondé sur des considérations probabilistes des objets sur lesquelles nous n'avons pas à nous étendre dans cet ouvrage. On pourra consulter par exemple [Ler81].

15.1.7 Le choix du nombre de classes

Combien y a-t-il réellement de classes dans les données ? En classification hiérarchique, ce nombre semble dépendre de l'observateur des données, puisque celui-ci doit sélectionner une valeur de coupure. Il est cependant possible de trouver des critères permettant une détermination automatique de cette valeur.

Certaines heuristiques simples peuvent être appliquées en observant uniquement la suite des valeurs de l'indice. En général, on préfère utiliser une mesure qui tient compte de la variance des données dans chaque classe. Notons δ_i^j , le symbole de Kronecker, qui vaut 1 quand $i = j$ et 0 sinon, C le nombre de classes et μ_j le centre de gravité de la classe j . On définit la quantité suivante :

$$T = \frac{1}{m} \sum_{j=1,C} \sum_{i=1,m} \delta_i^j |\mathbf{x}_i - \mu_j|^2$$

T vaut donc $\frac{1}{m}$ fois la somme sur toutes les classes de la distance de tous les points de cette classe au centre de gravité de cette classe. On appelle souvent T la *somme des variances intra classes*.

On peut démontrer que T diminue quand le nombre de classes C augmente, donc que cette valeur varie en sens inverse de l'indice de la hiérarchie.

Une comparaison faite entre diverses heuristiques [MC85] semble indiquer qu'une bonne valeur de compromis pour l'indice est celle qui maximise la quantité :

$$\frac{1}{T} \frac{m - C}{C - 1} \tag{15.1}$$

Il existe aussi des critères statistiques non paramétriques, comme celui proposé par Lerman ([Ler81]), et d'autres formules dans le même esprit que celle proposée ci-dessus, comme celle de Akaike et Schwarz ([JW98]).

15.2 La classification non hiérarchique de données numériques

15.2.1 La méthode des k -moyennes

Webb [Web99] donne un exemple que nous reprenons ici pour présenter cette méthode. Soient six objets numérotés de 1 à 6 dans \mathbb{R}^2 , comme sur la figure 15.3. On cherche à les regrouper en deux classes, autrement dit à en faire une partition à deux blocs. Pour cela, on commence par en tirer deux au hasard, disons les points 5 et 6.

4. Cette méthode utilise à la place de la distance euclidienne une distance particulière, appelée « dissimilarité informationnelle ».

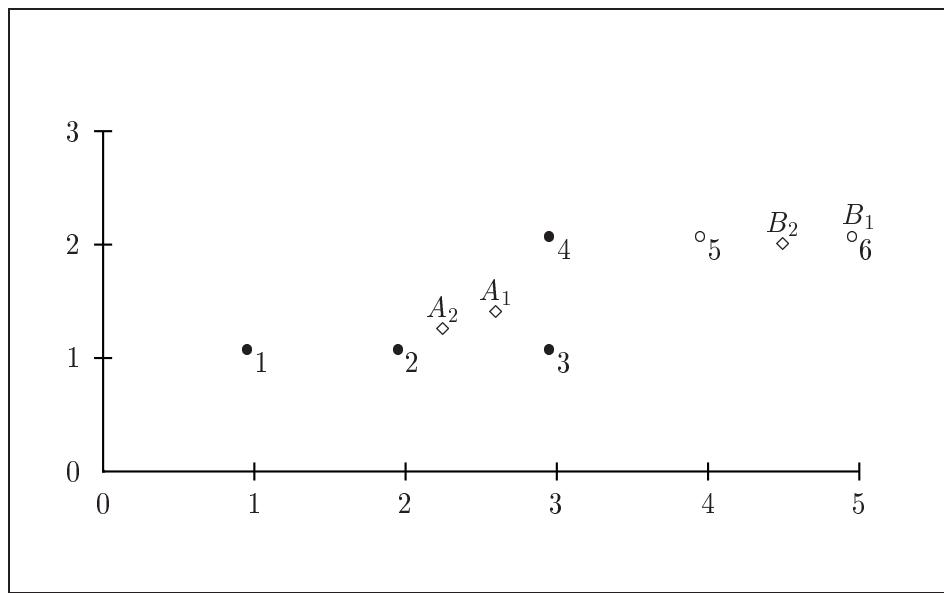


FIG. 15.3 – Un ensemble de points à partitionner à deux classes et le résultat de la méthode des 2-moyennes en partant des points 5 et 6.

Une première phase va allouer les six points aux deux classes sur le critère suivant: ceux qui sont plus près (au sens de la distance euclidienne) du point 5 que du point 6 sont dans la première classe, les autres dans la seconde. La première classe est donc pour le moment le bloc (1, 2, 3, 4, 5) et la seconde le bloc (6). La seconde phase consiste à calculer le centre de gravité de ces deux classes. Le premier, appelé A_1 a pour coordonnées (1.6, 2.4), le second, B_1 est le point 6, aux coordonnées (5, 2).

On recommence les deux phases : les six points sont alloués aux deux classes en fonction de A_1 et de B_1 . On obtient les blocs (1, 2, 3, 4) et (5, 6). Les centres de gravité de ces deux blocs sont A_2 et B_2 , aux coordonnées (2.25, 1.25) et (4.5, 2).

Une nouvelle passe ne change plus la partition : l'algorithme a convergé.

Il est intéressant de calculer une qualité globale de la classification. Généralement, on utilise ici aussi T , la somme des variances intra classes. Les valeurs successives de ce critère sur les partitions obtenues sont les suivantes :

$$\begin{aligned} (1,2,3,4,5), (6) & \quad 6.4 \\ (1,2,3,4), (5,6) & \quad 4.0 \end{aligned}$$

On peut montrer que l'algorithme des k -moyennes fait en effet diminuer la valeur T , mais rien n'assure que le minimum global soit atteint : la convergence peut en effet mener à un minimum local.

Si on initialise l'algorithme avec les points 2 et 5, la convergence vers la partition (1, 2, 3), (4, 5, 6) est réalisée d'entrée pour la valeur $T = 3.3$ du critère. La première initialisation, avec les points 5 et 6, converge donc vers un minimum local.

L'algorithme des k -moyennes est un cas particulier des algorithmes de classification par *allocation-recentrage*. La phase d'allocation est ici le calcul des classes à partir des centres de gravité provisoires, et la phase de recentrage le calcul des nouveaux centres de gravité des classes que l'on vient d'établir. Cette technique peut se voir de manière encore plus générale comme une application particulière de l'algorithme *EM*, qui sera abordé au paragraphe suivant.

Il existe une grande variété d'algorithmes du type k -moyennes, permettant en particulier de

faire naître et mourir des classes au cours des calculs, ce qui donne de la souplesse à la méthode : son inconvénient évident est en effet que l'utilisateur doit donner au départ le nombre de classes, ce qui n'est pas très réaliste.

Il est aussi possible de choisir le nombre de classes en faisant plusieurs essais avec des valeurs différentes et en calculant pour chacune une valeur du type de celle proposée pour les méthodes hiérarchiques à l'équation 15.1.

15.2.2 L'estimation d'une somme pondérée de distributions gaussiennes

On peut faire une analogie raisonnée entre la méthode des k -moyennes et la technique non paramétrique du plus proche voisin (chapitre 14, paragraphe 14.3.3). De même, il est possible de faire des hypothèses paramétriques sur la distribution des objets. Souvent, on considère qu'ils sont des tirages i.i.d. d'une distribution multigaussienne, encore appelée un mélange de gaussiennes (de lois normales) (*mixture of normal distributions model*), c'est-à-dire une somme pondérée de gaussiennes de moyennes et de matrices de covariance inconnues. En pratique, cela revient à supposer que chaque classe est une gaussienne avec ses caractéristiques particulières et une probabilité *a priori* plus ou moins forte⁵.

Nous avons donné au chapitre 14 la formule d'une distribution gaussienne dans \mathbb{R}^d . Nous la rappelons ici en notant $p(\mathcal{N}(\boldsymbol{\mu}, Q))$ la densité de cette distribution, avec $\boldsymbol{\mu}$ sa moyenne et Q sa matrice de covariance.

$$p(\mathcal{N}(\boldsymbol{\mu}, Q)) = \frac{|Q|^{-1/2}}{(2\pi)^{d/2}} \exp \left\{ -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T Q^{-1}(\mathbf{x} - \boldsymbol{\mu}) \right\} \quad (15.2)$$

Un mélange de C gaussiennes s'écrit donc :

$$p(\mathcal{N}(\boldsymbol{\mu}, Q)) = \sum_{j=1,C} k_j \frac{|Q_i|^{-1/2}}{(2\pi)^{d/2}} \exp \left\{ -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T Q_i^{-1}(\mathbf{x} - \boldsymbol{\mu}_i) \right\} \quad (15.3)$$

avec

$$\sum_{j=1,C} k_i = 1$$

En supposant C connu, il faut estimer les C scalaires k_i , les C matrices symétriques Q_i de dimension $d \times d$ et les C vecteurs $\boldsymbol{\mu}_i$ de dimension d pour tout i entre 1 et C , soit $\frac{d^2+2d+2}{2}$ paramètres.

On utilise pour cela l'algorithme *EM*, que nous avons déjà rencontré au chapitre 13 pour l'apprentissage des HMM. Une explication de la méthode *EM* et de son application à l'estimation des paramètres des mélanges de gaussiennes est donnée en annexe 18.9. Cette méthode souffre aussi de ce que C doit être fixé à l'avance. Là encore, diverses techniques sont applicables pour trouver la meilleure valeur, mais on peut s'appuyer ici sur l'hypothèse paramétrique faite sur la distribution des objets. Une technique éprouvée est de maximiser le « critère bayésien d'information » (*BIC*), comme présenté dans [JW98].

15.2.3 Un exemple

La figure 15.4 illustre les performances de cette méthode par comparaison avec celle des k -moyennes. On y voit 600 points, 287 tirés selon une première loi normale de moyenne $\boldsymbol{\mu}_1^T =$

5. Sous ce point de vue l'algorithme des k -moyennes revient à faire l'hypothèse que les classes proviennent de distributions gaussiennes ayant toutes la même matrice de covariance ; les probabilités *a priori* sont quelconques.

$(1 \ 1)$ et de matrice de covariance $Q_1 = \begin{pmatrix} 0.1 & 0.0 \\ 0.0 & 0.1 \end{pmatrix}$ et 313 selon une seconde de moyenne $\mu_1^T = (0.8 \ 0.8)$ et de matrice de covariance $\begin{pmatrix} 0.02 & 0 \\ 0 & 0.02 \end{pmatrix}$. Les deux distributions sont supposées équiprobables. Les surfaces d'équidensité de ces distributions sont les cercles en trait plein; la probabilité qu'un point soit tiré à l'intérieur est de 90 %. L'erreur bayésienne (le mieux que l'on puisse faire, voir le chapitre 14) vaut ici 13 %.

Les programmes qui implantent l'algorithme *EM* et celui des k -moyennes, connaissent les 400 points et le fait qu'il y ait deux classes à trouver, mais ils ignorent la classe de chaque point et la probabilité *a priori* des classes.

Sur la figure de gauche est donné le résultat de *EM*. Il trouve deux distributions gaussiennes assez proches des originales. Une probabilité *a priori* de 0.57 est attribuée à la première classe et de 0.43 à la seconde. Les distributions sont représentées par les ellipses en pointillés, qui correspondent aux valeurs suivantes :

$$\hat{\mu}_1^T = (1.043 \ 1.018) \quad \hat{Q}_1 = \begin{pmatrix} 0.097 & -0.005 \\ -0.005 & -0.093 \end{pmatrix}$$

$$\hat{\mu}_2^T = (0.792 \ 0.808) \quad \hat{Q}_2 = \begin{pmatrix} 0.025 & 0.002 \\ 0.002 & 0.019 \end{pmatrix}$$

La matrice de confusion de la classification des données initiales est la suivante; elle donne une erreur de 27 % :

	classe 1	classe 2
classe 1	199	88
classe 2	19	294

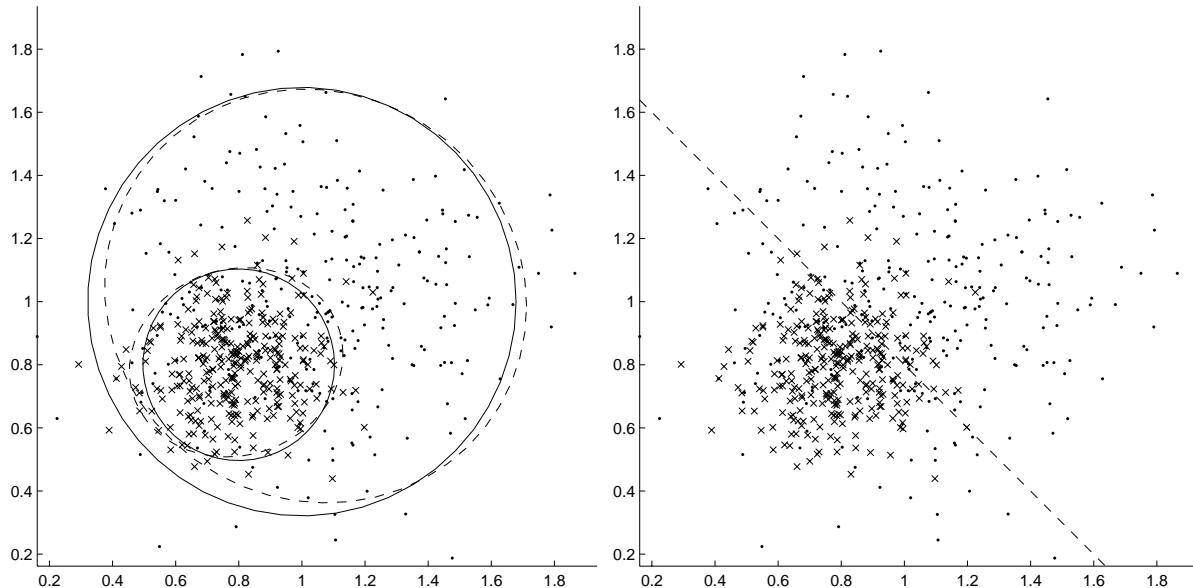


FIG. 15.4 – L'algorithme *EM* retrouve un mélange de distributions gaussiennes, mais l'algorithme des 2-moyennes est plus en difficulté.

Sur la figure de droite, l'algorithme des 2-moyennes sépare les points en deux classes. La surface séparatrice est la droite médiatrice des centres de gravité des classes trouvées (en pointillé). La

matrice de confusion de la classification des données initiales est la suivante (ce qui donne une erreur de 34 %):

	classe 1	classe 2
classe 1	162	125
classe 2	10	308

15.3 La classification de données symboliques

15.3.1 Les données binaires et catégorielles

Quand on quitte l'espace de représentation \mathbb{R}^d , de nouveaux problèmes apparaissent pour la classification automatique. Prenons d'abord le cas des données binaires, dans lequel chaque exemple objet est un vecteur de \mathbb{B}^d composé de d bits. Quelles mesures de distance peut-on proposer dans une telle représentation ?

La distance la plus simple est celle de Hamming qui mesure le nombre de bits différents entre deux objets, divisé par d . Sa valeur est donc toujours comprise entre 0 et 1. Mais on peut proposer d'autres mesures qui ne vérifient pas forcément les propriétés de la distance⁶. Soient deux objets \mathbf{x} et \mathbf{y} de \mathbb{B}^d . Notons :

- a le nombre d'attributs qui sont *VRAI* pour \mathbf{x} et \mathbf{y} ;
- b le nombre de ceux qui sont *VRAI* pour \mathbf{x} et *FAUX* pour \mathbf{y} ;
- c le nombre de ceux qui sont *FAUX* pour \mathbf{x} et *VRAI* pour \mathbf{y} ;
- d le nombre d'attributs qui sont *FAUX* pour \mathbf{x} et \mathbf{y} .

Les mesures de dissimilarité suivantes entre \mathbf{x} et \mathbf{y} sont classiques :

Nom	Formule	Commentaire
Hamming	$\frac{c+d}{a+b+c+d}$	est une distance
Russel et Rao	$1 - \frac{a}{a+b+c+d}$	ne vérifie pas la première propriété de la distance
Jaccard	$1 - \frac{a}{a+b+c}$	est une distance
Yule	$1 - \frac{ad-bc}{ad+bc}$	

Il est difficile d'appliquer l'algorithme non hiérarchique des k -moyennes à des données binaires, car il fait appel à la notion de centre de gravité *a priori* intraduisible dans \mathbb{B}^d . Certains auteurs ont proposé des techniques empiriques pour contourner la situation. Nous en verrons

6. Rappelons la définition d'une distance, donnée en 3.2 : Une distance Δ sur un espace $E \times E$ est une application de $E \times E$ dans \mathbb{R}^+ si et seulement si elle vérifie les propriétés :

- $\Delta(x, y) = 0 \iff x = y$;
- $\forall x, y \in \Sigma, \Delta(x, y) = \Delta(y, x)$ (symétrie) ;
- $\forall x, y, z \in \Sigma \quad \Delta(x, y) \leq \Delta(x, z) + \Delta(z, y)$ (inégalité triangulaire).

un exemple au paragraphe suivant (dans le cas de la représentation attribut-valeur) qui peut s'appliquer en particulier à celui de la logique des propositions.

En revanche, les algorithmes hiérarchiques peuvent toujours s'appliquer, à condition de savoir calculer un indice ultramétrique entre deux ensembles d'objets binaires. Les indices du lien simple, de Ward et de Lerman, sont en particulier calculables, car ils ne font pas appel au centre de gravité des ensembles. Nous verrons aussi au chapitre 17 que l'on ne peut pas mesurer la ressemblance entre deux objets binaires avec le nombre de propositions logiques qui ne valent par exemple *VRAI* sur les deux que si on limite volontairement l'ensemble des propositions disponibles par un biais (par exemple, si on décide de n'y mettre que les conjonctions).

15.3.2 Les attributs nominaux : la représentation attribut-valeur

Dans le cas où les attributs sont catégoriels (nominaux), un grand nombre de méthodes hiérarchiques ont été proposées. Elles sont fondées sur le calcul d'une distance entre objets catégoriels, puis sur son extension à un indice ultramétrique. Prenons l'exemple suivant. Un oiseau est défini par trois attributs :

- le fait que son bec soit *aplati* ou non ;
- sa *taille* qui peut prendre trois valeurs : *petite*, *moyenne* ou *grande* ;
- la *couleur* de son cou qui peut prendre quatre valeurs *roux*, *orange*, *gris* ou *noir*.

Soient deux oiseaux définis par :

	aplati	taille	couleur	nom
\boldsymbol{x}_1	<i>VRAI</i>	moyenne	roux	nette rousse
\boldsymbol{x}_2	<i>FAUX</i>	moyenne	noir	corneille noire

On peut par exemple généraliser la distance de Hamming par le calcul suivant⁷ :

$$\Delta(\boldsymbol{x}_1, \boldsymbol{x}_2) = 1 - (1 + 0 + 3) = 1 - \frac{1}{3}(2\frac{1}{2} + 3\frac{0}{3} + 4\frac{1}{4}) = \frac{1}{3}$$

Cette formule peut aussi être remplacée par la suivante, qui considère que *aplati* = *VRAI* et *aplati* = *FAUX* sont en quelque sorte deux fois moins différents que *couleur* = *roux* et *couleur* = *noir*, puisqu'il y a deux modalités pour la première variable et quatre pour la seconde :

$$\Delta(\boldsymbol{x}_1, \boldsymbol{x}_2) = 1 - \frac{1}{3}(\frac{1}{2} + \frac{0}{3} + \frac{1}{4}) = \frac{1}{3} = 0.25$$

Cette dernière expression varie entre 0 et $\frac{1}{3}(\frac{1}{2} + \frac{1}{3} + \frac{1}{4}) = 0.36$, mais il est facile de la ramener entre 0 et 1 si nécessaire.

Il est intéressant de noter qu'un certain nombre de méthodes fondées sur les concepts de l'espace des versions (voir le chapitre 4) ont été proposées pour construire des classifications non hiérarchiques. On a remarqué plus haut que la méthode des k -moyennes n'est pas applicable, car la notion de centre de gravité n'existe plus. Mais il est possible de construire des algorithmes analogues comme *CLUSTER/2* ([MS83]) dont l'algorithme est schématiquement donné en 15.2.

Un certain nombre de commentaires sont nécessaires pour préciser cet algorithme.

- Le choix des amorces n'est fait au hasard que la première fois. Quand la partition trouvée est meilleure que la meilleure partition courante, on cherche à l'améliorer encore en choisissant des objets « au centre » de chaque bloc. Sinon, on cherche à la corriger en choisissant des objets « au bord » de chaque bloc.

7. Nous ne donnons pas la formule générale, qui n'ajoute rien à la compréhension.

Algorithme 15.2 Un algorithme de classification non hiérarchique pour données symboliques

Fixer le nombre de classes : k .

tant que Le test de fin n'est pas satisfait **faire**

Choisir k objets amorces.

pour chaque amorce **faire**

Apprendre un ensemble de concepts
discriminants vis-à-vis des autres amorces.

fin pour

Modifier les concepts pour en déduire
un ensemble de partitions sur les objets

Choisir la meilleure

fin tant que

- La qualité d'une partition est évaluée par un mélange de critères parmi lesquels on peut citer :
 1. l'adéquation d'un concept, c'est-à-dire le nombre d'objets qu'il peut couvrir par rapport au nombre d'objets qu'il couvre dans l'ensemble à classer ;
 2. la simplicité d'un concept, que l'on peut mesurer par le nombre d'attributs qui y sont présents ;
 3. l'intersection des concepts : elle est nulle dans l'ensemble des objets à classer, mais peut se mesurer dans l'absolu.
 4. etc.

15.3.3 Les données logiques

Que devient l'apprentissage non supervisé dans des données décrites par des structures symboliques complexes comme les formules de la logique des prédicts ou les arbres ? Ce sujet est difficile, puisque la notion de distance ou de similarité dans ces espaces de représentation n'est pas naturelle. Pourtant, il est la clé du développement de l'ECD pour la découverte de concepts évolués.

Il existe un certain nombre de travaux de conceptualisation et de réalisation dans ce domaine. Il faudrait introduire de nouveaux concepts et de nouvelles définitions pour aborder le sujet, même rapidement. Nous préférons donc ici renvoyer le lecteur à l'article de synthèse de G. Bisson dans [DKBM00] sur la notion de similarité dans les domaines statistiques et symboliques. C'est une excellente mise en lumière des problèmes et des solutions actuelles.

15.4 La découverte automatique

15.4.1 Présentation

Soit P la période d'une planète de notre système solaire (la durée de son « année »), R la longueur du grand axe de l'ellipse de son orbite autour du soleil, D sa distance moyenne au soleil et d sa densité.

Si P est mesurée en jours terrestres et D et R en millions de kilomètres, on peut donner à un programme de découverte automatique les mesures suivantes comme exemples (les valeurs sont approximatives) :

Planète	P	R	D	d
Vénus	224	220	60	5.6
Terre	365	300	150	5.5
Mars	686	420	230	3.9
Jupiter	4400	1500	780	1.3
Saturne	10750	2800	1400	0.7
Neptune	60140	9000	4500	1.8
Pluton	90400	25000	6000	2.1

Ce programme devra fournir en sortie la loi de Kepler :

$$P^2 = cR^3 \quad \text{avec} \quad c \sim 5 \cdot 10^{-3}$$

et “découvrir” aussi que les attributs R et d ne sont pas reliés simplement aux deux premiers⁸.

Remarquons dans cet exemple la différence entre la découverte automatique et une simple technique de régression. L'utilisation de cette dernière pourrait par exemple produire un polynôme de degré donné des variables donnant une bonne approximation des données d'apprentissage, mais elle aurait deux défauts : d'abord l'expression analytique serait plus complexe, en forçant D et d à intervenir ; d'autre part, rien n'assure que de nouvelles données (celles d'Uranus, par exemple) issues de la même loi répondraient à l'équation polynomiale. En d'autres termes, la régression fournirait un apprentissage par cœur.

On cherche donc ici à découvrir une véritable généralisation des données, sous l'hypothèse générale de la *simplicité* de la loi qui lie les variables.

La meilleure façon de présenter les concepts de ces méthodes est de présenter un système classique de découverte automatique: BACON ([Lan96]).

15.4.2 La découverte de fonctions simples

Le système BACON est né en 1978 et n'a pas cessé d'évoluer depuis, dans le but de découvrir des lois de forme de plus en plus complexe. Dans son principe, BACON n'est pas très différent d'un algorithme de recherche heuristique en espace d'états (voir le chapitre 3); c'est l'adéquation aux données et la simplicité qui sert de pilote à sa recherche.

L'algorithme fonctionne par création de nouveaux attributs à partir de ceux qui sont connus, jusqu'à en avoir fabriqué un dont la valeur est constante sur tous les exemples: la loi est alors trouvée.

15.4.2.1 Un exemple

Reprenons l'exemple ci-dessus, avec pour seules données les valeurs des attributs P et R . Le fonctionnement idéal de BACON serait dans ce cas le suivant :

Recherche d'une relation $P = aR + b$. Pour cela, l'algorithme construit par régression linéaire la droite de coefficients a et b qui minimise la somme des carrés des distances aux exemples.

Si cette distance est trop importante, comme c'est le cas ici, on abandonne cette hypothèse de loi linéaire.

8. P et R pourraient être reliés si l'excentricité de l'ellipse était donnée; d n'a pas de corrélation aussi directe avec les autres attributs.

Création d'un nouvel attribut X_1 . Puisque P et R croissent ensemble, on crée l'attribut

$$X_1 = \frac{R}{P}$$

et on le calcule sur les exemples. Dans le cas où les deux variables varient en sens inverse, on crée leur produit.

Création d'un nouvel attribut X_2 . Sur les trois variables P , R et X_1 , on applique le même argument qu'aux deux étapes précédentes. Après avoir vérifié qu'il n'existe pas de relation linéaire entre les trois variables et constaté que R et X_1 varient en sens inverse, on crée la variable :

$$X_2 = RX_1 = \frac{R^2}{P}$$

Ici intervient évidemment un élagage dans le graphe que l'on est en train de développer : on aurait pu créer et évaluer la linéarité de la variable $R.P/P$, mais un calcul formel simple montre qu'il est inutile de calculer sa valeur sur les données, puisqu'elle est égale à P .

Création d'un nouvel attribut X_3 et conclusion. En continuant ce procédé avec quatre variables, on constate que X_2 et X_1 varient en sens inverse. On va donc créer :

$$X_3 = X_2.X_1 = R^3/P^2$$

qui prend une valeur quasiment constante (égale à c) sur l'ensemble des données. On peut donc considérer avoir découvert la loi de Kepler $P^2 = c.R^3$.

15.4.2.2 Complexité de calcul

Sans décrire complètement l'algorithme pour en faire l'analyse en complexité, il est clair que le procédé décrit est de nature exponentielle en fonction du nombre de données. Il est naturel d'utiliser les techniques d'élagage de la programmation combinatoire (voir le chapitre 3) pour l'implanter, y compris les techniques heuristiques de l'intelligence artificielle. Nous ne développerons pas ce sujet ; remarquons simplement qu'il est évidemment indispensable d'utiliser au mieux les éventuelles connaissances *a priori* sur le domaine.

15.4.3 Découverte de lois plus complexes

15.4.3.1 Lois polynomiales

Bien que déjà d'une grande complexité calculatoire, le procédé de base décrit n'est pas encore suffisant pour découvrir certaines lois. Par exemple, il aurait été impossible de découvrir une relation polynomiale simple comme :

$$R = aP^2 + b$$

Les versions ultérieures de BACON ont intégré la recherche de telles lois en utilisant la technique suivante : on cherche s'il existe une dérivée d'ordre p d'une variable par rapport à une autre qui soit constante sur les exemples. Si oui, ceci donne l'ordre $p + 1$ du polynôme qui les lie ; ses coefficients sont alors calculables assez facilement.

Par exemple, pour les variables X et Y dont on a les exemples ci-dessous, on peut calculer certaines approximations des dérivées premières et secondes de Y par rapport à X .

i	$X(i)$	$Y(i)$	$\frac{\partial Y}{\partial X}$	$\frac{\partial^2 Y}{\partial X^2}$
1	1	6		
2	3	34	4	
3	6	121	29	3
4	10	321	50	3
5	15	706	77	3

On a en effet par exemple, pour la troisième ligne :

$$\frac{\partial Y}{\partial X} \simeq \frac{Y(3) - Y(2)}{X(3) - X(2)} = (121 - 34)/(6 - 3) = 29$$

Dans cet exemple, on constate que la dérivée seconde de Y par rapport à X est constante sur les données et égale à 3. On en déduit l'existence de la loi :

$$Y = 3X^2 + bX + c$$

Les valeurs des constantes b et c peuvent être obtenues à partir des exemples en calculant la variable auxiliaire :

$$X_1 = Y - 3X^2$$

puis en cherchant par régression la relation linéaire qui lie X_1 à X .

15.4.3.2 Lois trigonométriques

Pour augmenter le domaine de découverte, on peut aussi introduire les opérateurs trigonométriques, souvent utiles dans l'expression de lois régissant des constantes physiques.

15.4.4 Traitement des données bruitées

15.4.4.1 Le problème

Tout système du type de BACON doit trouver un réglage entre des possibilités contradictoires : si on admet par hypothèse que les données ne sont pas ou presque pas bruitées, il est possible d'introduire un assez grand nombre d'opérations élémentaires de découverte, comme celles données ci-dessus. En effet, seule une petite fraction des lois qu'il est possible de calculer en combinant ces opérations satisfairont assez exactement les exemples ; en revanche, si on veut tolérer plus de bruit, il faut laisser moins de possibilités combinatoires.

15.4.4.2 Exemple

L'exemple suivant illustre ce dilemme ; supposons avoir à notre disposition l'ensemble des possibilités de calcul évoquées et les données :

X	Y
0	0
2	0.3491
4	0.6981
6	1.0472

Un grand nombre de lois peut être découvert si on tolère ne serait-ce que 1 % de variation sur les valeurs de X et de Y . Par exemple la simple droite :

$$Y = 0.175 \times X$$

est une loi correcte de ce point de vue, car tous les exemples la vérifient pour des valeurs comprises entre 99 % et 101 % de celles données dans la table. On pourrait ainsi trouver un grand nombre d'exemples. Celui-ci maximise un critère de simplicité.

En revanche, si on suppose les valeurs d'apprentissage exactes à 10^{-6} , seules deux lois seront découvertes :

$$Y = 100 \times \sin(X/100)$$

(c'est à partir de celle-ci que le tableau des données a été calculé) et :

$$Y = \alpha X^3 + \beta X^2 + \gamma X$$

avec : $\alpha = -0.001504$, $\beta = 0.006325$ et $\gamma = 0.336$

On peut en effet faire passer exactement un polynôme de degré p par $p+1$ points ; d'autre part, BACON s'arrête dès qu'il a trouvé une loi le satisfaisant et procède par puissances croissantes dans la découverte des polynômes : il ne cherchera donc pas dans ce cas dans les degrés supérieurs à 4 en X .

15.4.4.3 Heuristiques

Diverses techniques peuvent être employées pour traiter ce problème. Outre des considérations purement numériques (calcul des intervalles d'erreur sur les variables intermédiaires créées par le programme, estimation de la variance du bruit dans les données en fonction des lois découvertes), BACON priviliege, selon son principe de base, la simplicité. Ainsi, à mesure d'erreur égale sur les exemples, une loi comportant moins de termes qu'une autre sera considérée comme meilleure.

15.4.5 Découverte de lois avec plus de deux variables

Nous n'avons parlé jusqu'ici que de la découverte de lois liant deux variables. Comment étendre la recherche au cas de formules plus complexes, comme pour découvrir la loi des gaz parfaits

$$PV = kN(T - 273)$$

avec P la pression d'une certaine quantité de gaz, V son volume, T sa température en degrés Celsius et N la quantité de gaz (en moles) ?

Idéalement, BACON fonctionne alors comme suit : il fixe N à (disons) 1, T à (disons) 10 et examine les variations des valeurs des deux variables restantes P et V pour les données où N et T valent 1 et 10.

Supposons donc que, parmi tous ceux dont on dispose, les exemples pour lesquels on a les contraintes $N = 1$ et $T = 10$ soient les suivants :

N	T	P	V
1	10	1000	2.36
1	10	2000	1.18
1	10	3000	0.78

Le module élémentaire de découverte décrit plus haut produira la loi:

$$V^{-1} = 4.25 \times 10^{-4} \cdot P$$

La variable intermédiaire X_1 est créée ; elle vaut 4.25×10^{-4} pour les trois exemples cités. On la calcule sur les autres exemples de la base de données, ce qui permet par exemple de constater qu'elle n'est pas constante : sa valeur restera alors pour le moment indéterminée pour ces autres exemples.

Ensuite, N est conservé à 1, mais la valeur de T est changée à (disons) 20, ce qui permet d'examiner d'autres exemples :

N	T	P	V
1	20	1000	2.44
1	20	2000	1.22
1	20	3000	0.81

Une loi linéaire est également découverte, mais la valeur de X_1 sera $4.1 \cdot 10^{-4}$ sur les trois exemples ci-dessus.

De même, pour $N = 1$ et $T = 30$, toujours en supposant la base de données assez complète, on pourra trouver des exemples permettant d'affirmer la loi linéaire :

$$V^{-1} = 3.96 \times 10^{-4} P$$

ce qui permettra de compléter les valeurs de X_1 .

Ceci fait, le programme va chercher une loi liant X_1 et les autres variables partout où la première a été calculée. Il trouvera la relation linéaire :

$$1/X_1 = 8.32 \times T + 2271$$

Ceci permet de définir les variables X_2 et X_3 , dont les valeurs 8.32 et 2271 sont rajoutées sur les exemples examinés, et sur eux seuls.

Le programme passe ensuite à $N = 2$ en procédant de la même manière. Il va induire la relation

$$1/X_1 = 16.64 \times T + 4542$$

et stocker de même à leur place les deux nouvelles valeurs de X_2 et X_3 ainsi trouvées. De même, pour $N = 3$, il ajoutera les valeurs 24.96 et 6814 aux exemples concernés.

On a alors assez calculé de données pour chercher une relation entre X_2 , X_3 et les variables de départ. On trouve ainsi :

$$X_2 = 8.32 \times N \quad \text{et} \quad X_3 = 2271 \times N$$

Les variables X_4 et X_5 , valant 8.32 et 2271 sur tous les exemples examinés sont alors créées. Leur calcul sur l'ensemble des autres exemples montre cette fois que ce sont en fait des constantes. Selon son principe général de fonctionnement, BACON remonte alors la suite des variables intermédiaires pour écrire la loi recherchée.

En suivant ce principe, le nombre d'exemples et la taille de l'espace de recherche croissent exponentiellement avec le nombre de variables ; de plus, l'algorithme doit effectuer des choix sur les valeurs pertinentes des variables. Ce dernier point peut être résolu soit à l'examen de l'ensemble des exemples, soit par un “oracle” supplémentaire aux seules données d'apprentissage.

15.4.6 Améliorations ultérieures

Un grand nombre d'heuristiques complémentaires ont été implantées dans BACON pour pallier l'exponentialité intrinsèque de sa démarche. Par exemple, l'hypothèse qu'il existe une symétrie dans la loi à découvrir permet d'élaguer la recherche.

15.5 La découverte non supervisée d'associations complexes d'attributs

Les méthodes décrites dans ce paragraphe constituent un complément récent aux méthodes statistiques classiques de détection de relations entre attributs. Elles ont été développées dans le cadre de la fouille de données. Elles traitent des données booléennes et non pas continues comme c'est le cas en statistique et s'intéressent au développement d'algorithmes performants prenant en compte la nature de ces données. Ces méthodes cherchent à repérer des implications logiques entre attributs ou ensembles d'attributs en utilisant la table de vérité de l'implication logique :

x	y	$x \Rightarrow y$
VRAI	VRAI	VRAI
FAUX	VRAI	VRAI
FAUX	FAUX	VRAI
VRAI	FAUX	FAUX

15.5.1 Les associations d'attributs binaires : définitions

Nous utilisons dans ce paragraphe le vocabulaire des bases de données : un exemple e (non supervisé) est appelé un *enregistrement*. Il est décrit par d attributs binaires (x_1, x_2, \dots, x_d), ou *champs*. La valeur à 1 d'un attribut pour un exemple est appelée un *item*.

Nous utiliserons comme exemple dans ce paragraphe la base de données (l'ensemble d'apprentissage) suivante, composée de dix exemples décrits par cinq attributs binaires :

	x_1	x_2	x_3	x_4	x_5
e_1	0	1	0	0	1
e_2	0	0	1	0	1
e_3	0	0	1	0	0
e_4	1	1	1	1	1
e_5	1	1	1	1	1
e_6	1	1	1	1	0
e_7	1	0	1	1	0
e_8	1	0	1	1	0
e_9	1	0	0	0	1
e_{10}	1	0	0	0	1

Définition 15.1

On appelle couverture (ou support) de $x_1 \Rightarrow x_2$ la probabilité $P(x_1, x_2)$ que x_1 et x_2 soient VRAI en même temps. Comme $P(x_1, x_2) = P(x_2, x_1)$, la couverture de $x_1 \Rightarrow x_2$ est la même que celle de $x_2 \Rightarrow x_1$.

Dans un tableau de données, on estime $P(x_1, x_2)$ par la fréquence d'observation : le nombre de fois où x_1 et x_2 sont VRAI ensemble, divisé par le nombre total d'enregistrements.

Dans notre exemple, la couverture de $x_1 \Rightarrow x_2$ vaut $P(x_1, x_2) = \frac{3}{10}$

Il est à remarquer que les implications à trop forte couverture sont inintéressantes puisqu'elles représentent le fait que deux assertions sont (presque) toujours vraies ensemble. Inversement, si la couverture est trop faible et si $P(x_1, x_2)$ est de l'ordre de grandeur de la probabilité du bruit, alors $x_1 \Rightarrow x_2$ peut n'être due qu'au bruit.

La couverture présente une propriété intéressante, celle que $P(x_1, x_2, \dots, x_k)$ est toujours plus grand que $P(x_1, x_2, \dots, x_k, x_{k+1})$ puisque le nombre de fois où x_1, x_2, \dots, x_k sont VRAI ensemble est toujours plus petit que celui où x_1, x_2, x_k et x_{k+1} sont VRAI ensemble. Cette propriété est la base de l'algorithme « A Priori » que nous verrons au paragraphe suivant.

Définition 15.2

On appelle confiance de $x_1 \Rightarrow x_2$ la probabilité conditionnelle de rencontrer x_2 à VRAI quand on a rencontré x_1 à VRAI, soit $P(x_2|x_1) = P(x_1, x_2)/P(x_1)$. La confiance de $x_2 \Rightarrow x_1$ est donc égale à $P(x_1, x_2)/P(x_2)$.

Dans notre exemple, la confiance de $x_1 \Rightarrow x_2$ vaut $\frac{P(x_1, x_2)}{P(x_1)} = \frac{3}{7}$, celle de $x_2 \Rightarrow x_1$ vaut $\frac{3}{4}$.

Définition 15.3

On appelle dépendance de x_1 et x_2 la valeur⁹ : $|P(x_2|x_1) - P(x_2)|$ définie quand $P(x_1) \neq 0$.

Dans notre exemple, la dépendance de x_1 et x_2 vaut $|\frac{3}{7} - \frac{4}{10}| = 0.03$. La dépendance de x_2 et x_1 vaut $|\frac{3}{4} - \frac{5}{10}| = 0.25$.

Définition 15.4

On dit que la dépendance entre x_3 et x_2 est fortuite (spurious) s'il existe un x_1 tel que $P(x_2|x_1, x_3) = P(x_2|x_1)$ ce qui s'exprime en disant que « x_2 est indépendant de x_3 pour un x_1 donné ».

C'est le cas pour notre exemple : $P(x_2|x_1, x_3) = P(x_2|x_1) = \frac{3}{10}$.

Les dépendances fortuites entre x_3 et x_2 représentent des associations du type $x_2 \Rightarrow x_3$ ou $x_3 \Rightarrow x_2$. Plus précisément, x_2 et x_3 sont fortuitement dépendants quand ils ont une cause commune qui « explique » leur corrélation fortuite. Cependant, x_2 et x_3 sont VRAI ensemble très souvent et présentent une couverture du même ordre que $x_1 \Rightarrow x_2$ et $x_1 \Rightarrow x_3$. L'immense majorité des systèmes de détection d'associations ne prend pas ce phénomène en compte et présente comme également valides les trois implications, $x_1 \Rightarrow x_2$, $x_1 \Rightarrow x_3$, $x_2 \Rightarrow x_3$, bien que la troisième soit fausse en un certain sens. L'approche des réseaux bayésiens (chapitre 12) est la seule qui tente de rendre compte de ce genre de phénomène.

15.5.2 L'apprentissage des associations

On a donc un tableau dont les colonnes sont les champs de la base de données (les attributs), les lignes sont les enregistrements de la base de données (les exemples) et chaque valeur à 1 d'un champ pour un enregistrement donné est appelé un *item*. Un ensemble d'items pour un enregistrement a été appelé un « itemset » par les inventeurs américains de ces techniques et le terme est resté en français. Dans le vocabulaire qui nous est familier, un itemset est donc un sous-ensemble de tous les attributs binaires valant 1 pour un exemple donné.

9. Ce mot est trompeur : la relation de dépendance n'est pas symétrique. Elle exprime en réalité combien x_2 dépend de x_1 .

Dans notre exemple, $\{x_2, x_3\}$ et $\{x_1, x_4\}$ sont des itemsets de e_5 .

Définition 15.5

Une association est une implication disant que deux itemsets sont VRAI ensemble pour un nombre suffisant d'exemples. La couverture de l'association est calculée comme le nombre d'itemsets, divisé par le nombre total d'enregistrements (d'exemples). Quand une couverture est supérieure à une valeur $MinCouv$ fixée à l'avance par l'utilisateur, on dit que l'itemset constitué par cette intersection est fréquent.

Dans notre exemple, en prenant $MinCouv = 3$, les itemsets (x_1, x_3, x_5) et (x_1, x_3, x_4) sont fréquents avec pour valeurs d'association : $s_{x_1x_3x_5} = 3$ et $s_{x_1x_3x_4} = 5$.

$MinCouv$ caractérise la couverture minimale exigée par un utilisateur pour une règle d'association. Le problème que résoud l'algorithme « A Priori » est de trouver tous les itemsets fréquents, c'est-à-dire ceux qui ont une couverture plus grande que $MinCouv$.

L'intérêt d'utiliser la notion de couverture des itemsets est double.

- D'une part, cela permet de prévoir tout un groupe d'associations à couverture importante. Considérons par exemple l'itemset composé de l'intersection des quatre champs $\{x_1, x_2, x_3, x_4\}$. Si cette intersection est fréquente, alors toutes les règles d'association associées à ce quadruplet ont une couverture fréquente. Par exemple, $x_1 \wedge x_2 \Rightarrow x_3 \wedge x_4$, $x_1 \Rightarrow x_2 \wedge x_3 \wedge x_4$, $x_1 \wedge x_2 \wedge x_3 \Rightarrow x_4$, $x_1 \wedge x_4 \Rightarrow x_3 \wedge x_2$, ... sont toutes des règles d'association fréquentes parce que, par définition, leur couverture est celle de $\{x_1, x_2, x_3, x_4\}$.
- D'autre part, l'utilisation du fait que l'intersection de deux itemsets a une couverture inférieure ou égale à celle de chacun d'eux permet la construction d'algorithmes rapides, comme « A Priori », pour trouver tous les itemsets fréquents.

Algorithme 15.3 Algorithme A Priori

Créer L_1 , l'ensemble des 1-itemsets fréquents par une consultation de la base de données.

tant que le test d'arrêt n'est pas satisfait faire

Étape 1 Utiliser L_{k-1} pour produire C_k contenant les k -itemsets candidats.

NB. : Ceci se fait sans consulter la base de données.

Étape 2 Ne conserver que les itemsets de C_k qui sont fréquents : ils constituent L_k .

NB. : Ceci demande une consultation de la base de données.

fin tant que

Exemple

Soit $(\{x_1, x_2, x_3\}, s_{x_1x_2x_3})$ le 3-itemset composé des champs x_1 , x_2 et x_3 , de couverture $s_{x_1x_2x_3} = 0.3$. À la troisième étape du traitement de notre exemple, on a :

$$\begin{aligned} L_3 = \{ & (\{x_1, x_2, x_3\}, s_{x_1x_2x_3}), \\ & (\{x_1, x_2, x_4\}, s_{x_1x_2x_4}), \\ & (\{x_1, x_3, x_4\}, s_{x_1x_3x_4}), \\ & (\{x_1, x_3, x_5\}, s_{x_1x_3x_5}), \\ & (\{x_2, x_3, x_4\}, s_{x_2x_3x_4}) \} \end{aligned}$$

Il faut alors en principe créer les ensembles pouvant être construits à partir de L_3 pour construire C_4 .

$$C_4 = \{(\{x_1, x_2, x_3, x_4\}, 0.3), (\{x_1, x_3, x_4, x_5\}, 0.2), \dots\}$$

En réalité, la construction de tous ces itemsets n'est pas nécessaire : comme $\{x_1, x_4, x_5\}$ n'est pas dans L_3 , $\{x_1, x_3, x_4, x_5\}$ n'a pas à être examiné.

Dans notre exemple, seuls les sous-ensembles de $\{x_1, x_2, x_3, x_4\}$ sont fréquents. $\{x_1, x_2, x_3, x_4\}$ est donc le seul candidat possible à former C_4 .

On aura donc : $C_4 = \{ (\{x_1, x_2, x_3, x_4\}, s_{x_1x_2x_3x_4}) \}$.

Comme $s_{x_1x_2x_3x_4} = 3$ alors $L_4 = \{ (\{x_1, x_2, x_3, x_4\}, 3) \}$ si $MinCouv \leq 3$ sinon L_4 est vide et le processus s'arrête.

15.5.3 Découverte de suites temporelles dans les données

15.5.3.1 Représentation des connaissances et définition du problème

Le principe de cette découverte automatique consiste à utiliser une information particulière souvent présente dans les bases de données : la date d'arrivée de chaque enregistrement. On va en déduire une description des itemsets en suites ordonnées par le temps. Si l'itemset i se trouve avant l'itemset j dans la suite, cela signifiera que i est survenu avant j .

Un itemset peut être lui-même représenté par une suite ordonnée selon un indice arbitraire (par exemple le numéro d'identification de l'item, ou l'ordre alphabétique des noms des items). Mais cela n'a pas de signification temporelle : le fait que l'item i se trouve avant l'item j , tous les deux contenus dans l'itemset k , ne signifie pas que i précède j dans le temps. Tous les items contenus dans un même itemset sont donc considérés comme contemporains.

Supposons que les champs soient indexés par les entiers naturels. Alors, nous dirons que $(5, 1)$ représente l'itemset constitué par le fait que les champs 1 et 5 prennent la valeur *VRAI*. $(5, 1)$ est équivalent à $(1, 5)$. On dira qu'un itemset I_1 est *inclus* dans l'itemset I_2 quand tous les éléments de I_1 se trouvent dans I_2 . On le note par $I_1 \subseteq I_2$.

Suite temporelle d'itemsets

Par exemple, la suite d'itemsets $S = \langle(3, 6, 9), (5, 1)(7)\rangle$ indique que l'itemset $(3, 6, 9)$ précède temporellement l'itemset $(5, 1)$.

Définition 15.6

Un enregistrement est dit appartenir à la couverture d'une suite lorsque cette suite apparaît dans cet enregistrement. La probabilité $P(S)$ d'une suite S est estimée par le rapport du nombre d'enregistrements qui appartiennent à sa couverture, divisé par le nombre total d'enregistrements. On appelle couverture la valeur de $P(S)$.

Soit alors une mesure de fréquence M . On peut prendre pour M la couverture ou une autre mesure d'association comme la probabilité conditionnelle $P(S_1|S_2)$ d'observer une suite S_1 sachant qu'une autre S_2 a été observée. On dira qu'un événement est M -fréquent quand la fréquence de M est supérieure à une valeur fixée à l'avance par l'utilisateur. On peut alors définir l'inclusion de deux listes de plusieurs façons et chacune engendre un problème de découverte différent.

Définition 15.7

La suite $S_1 = \langle a_1, \dots, a_n \rangle$ est incluse dans la suite $S_2 = \langle b_1, \dots, b_m \rangle$, avec $m \leq n$, s'il existe une suite d'entiers $i_1 < i_2 < \dots < i_n$ telle que $a_1 \subseteq b_{i_1}, a_2 \subseteq b_{i_2}, \dots, a_n \subseteq b_{i_n}$.

Exemple

La suite $\langle(3, 6, 9), (5, 1)\rangle$ est incluse dans la suite $\langle(3, 7, 6, 9), (7, 3, 9), (1, 2, 5)\rangle$. Quand la longueur de suite S_1 est k , on dit qu'elle est une k -suite. Une suite de longueur 1, une 1-suite,

contient donc un seul itemset.

Définition 15.8

Le problème de la découverte automatique pour les suites temporelles d'itemsets s'énonce ainsi : trouver toutes les suites d'itemsets qui soient à la fois M -fréquentes et maximales.

15.5.3.2 Les quatre phases de la solution au problème

Phase 1 : création de la base.

Chaque enregistrement contient une suite d'items ordonnée dans le temps. Par exemple, si le consommateur j a acheté de la bière, du pain et de la mayonnaise au temps t , et de la mayonnaise, du pain, et des biscuits au temps $t + 1$, l'enregistrement j est alors : $\langle(\text{bière}, \text{pain}, \text{mayonnaise}), (\text{mayonnaise}, \text{pain}, \text{biscuits})\rangle$.

Phase 2 : détermination de tous les itemsets M -fréquents.

Ce sont par définition les 1-suites M -fréquentes. Ces itemsets fréquents sont renommés par un index entier pour faciliter les appariements.

Exemple

Si les produits « bière », « pain » et « mayonnaise » sont fréquemment achetés et si la M -fréquence est simplement la couverture, alors ces items sont fréquents. On remplace alors, par exemple, « bière » par « 1 », « pain » par « 2 », « mayonnaise » par « 3 ». Un item peut comprendre plusieurs produits à la fois, comme par exemple (« bière, pain »). Supposons qu'il soit aussi fréquent et qu'on lui associe le nombre « 4 ». Si « biscuits » n'est pas fréquemment acheté, alors aucun indice ne lui est associé, pas plus qu'à (« mayonnaise, pain, biscuits »).

Phase 3 : réécriture de la base.

Chaque enregistrement est transformé en l'ensemble des itemsets fréquents qu'il contient. Si un enregistrement ne contient aucun itemset fréquent, alors il est éliminé. Il n'intervient plus que dans le décompte du nombre total d'enregistrements.

Exemple

Considérons l'enregistrement :

$j =: \langle(\text{bière}, \text{pain}, \text{mayonnaise}), (\text{mayonnaise}, \text{pain}, \text{biscuits})\rangle$. Il est récrit comme :

$\langle(\text{ « bière »}, \text{ « pain »}, \text{ « mayonnaise »}), (\text{ « bière, pain »}), (\text{ « mayonnaise »}, \text{ « pain »})\rangle$

qui est finalement récrit :

$\langle(1, 2, 3, 4), (3, 2)\rangle$.

Phase 4 : trouver toutes les suites fréquentes, et ne conserver que les maximales.

Comme pour la détection des associations, on va engendrer tous les candidats de taille $k + 1$ à partir des séquences de taille k , puis on éliminera les candidats qui ne sont pas fréquents.

Exemple [AMS⁺95].

Soit la base de données :

e_1	$\langle(1, 5), (2), (3), (4)\rangle$
e_2	$\langle(1), (3), (4), (3, 5)\rangle$
e_3	$\langle(1), (2), (3), (4)\rangle$
e_4	$\langle(1), (3), (5)\rangle$
e_5	$\langle(4), (5)\rangle$

Posons qu'une suite est fréquente si sa couverture est supérieure ou égale à 2.

1-suite	couv.	2-suite	couv.	3-suite	couv.	4-suite	couv.	max. suite	couv.
$\langle 1 \rangle$	4	$\langle 12 \rangle$	2	$\langle 123 \rangle$	2	$\langle 1234 \rangle$	2	$\langle 1234 \rangle$	2
$\langle 2 \rangle$	2	$\langle 13 \rangle$	4	$\langle 124 \rangle$	2	$\langle 1345 \rangle$	1	$\langle 135 \rangle$	2
$\langle 3 \rangle$	4	$\langle 14 \rangle$	3	$\langle 134 \rangle$	3			$\langle 45 \rangle$	2
$\langle 4 \rangle$	4	$\langle 15 \rangle$	3	$\langle 135 \rangle$	2				
$\langle 5 \rangle$	4	$\langle 23 \rangle$	2	$\langle 145 \rangle$	1				
		$\langle 24 \rangle$	2	$\langle 234 \rangle$	2				
		$\langle 25 \rangle$	0	$\langle 235 \rangle$	0				
		$\langle 34 \rangle$	3	$\langle 245 \rangle$	0				
		$\langle 35 \rangle$	2	$\langle 345 \rangle$	1				
		$\langle 45 \rangle$	2						

Par exemple, on ne teste pas la suite $\langle 125 \rangle$ puisque $\langle 25 \rangle$ n'est pas fréquente. La suite $\langle 135 \rangle$ est la seule suite de longueur 3 qui soit fréquente et non contenue dans $\langle 1234 \rangle$.

15.5.3.3 Généralisation de la notion de suite « contenue dans une autre suite » en fonction des connaissances du domaine

La notion de base est celle de l'inclusion (définition 15.7). Cette notion va être généralisée selon le type des connaissances introduites.

Suite contenue dans une autre en présence d'une taxonomie de généralité

Définition 15.9

Soit T une taxonomie de généralité¹⁰. Un enregistrement contient un item x si x est dans T ou si un ancêtre de x est dans T . Un enregistrement contient un itemset y si tout item de y est contenu dans T .

La définition de l'inclusion peut alors être modifiée pour prendre en compte cette nouvelle notion de contenance :

Définition 15.10

On dira que la suite $S_1 = \langle a_1, \dots, a_n \rangle$ est incluse dans la suite $S_2 = \langle b_1, \dots, b_m \rangle$, $m \geq n$, s'il existe une suite d'entiers $i_1 < i_2 \dots < i_n$ telle que $a_1 \subseteq b_{i_1}, a_2 \subseteq b_{i_2}, \dots, a_n \subseteq b_{i_n}$, où maintenant \subseteq signifie « contenu dans » comme nous venons de le définir.

Exemple

Considérons les deux itemsets:

$$e_1 = \langle (1), (2), (3, 4) \rangle$$

$$e_2 = \langle (1, 2), (5), (3) \rangle$$

Cherchons les suites de couverture maximale communes aux deux enregistrements. On trouve $\langle (1)(3) \rangle$ et $\langle (2)(3) \rangle$.

Admettons alors que nous ayons la connaissance suivante :

- A est le parent de 2, 4 et 5
- B est le parent de 1 et 3.

En introduisant les relations de parenté dans les itemsets, ils deviennent :

$$e_1 = \langle (1, B), (2, A), (3, B, 4, A) \rangle,$$

10. Voir le chapitre 3 et le chapitre 11

$$e_2 = \langle (1, B, 2, A), (5, A), (3, B) \rangle$$

On obtient les itemsets fréquents suivants de longueur 2 :

$$\begin{aligned} & \langle (1)(3) \rangle, \langle (2)(3) \rangle, \\ & \langle (B)(B) \rangle, \langle (A)(B) \rangle, \langle (1)(B) \rangle, \\ & \langle (2)(B) \rangle, \langle (A)(3) \rangle, \langle (A)(3) \rangle, \end{aligned}$$

qui sont de simples généralisations des itemsets existants.

Mais on obtient aussi une suite de longueur 3 :

$$\langle (B)(A)(B) \rangle$$

dont sont fréquentes les instances $\langle (1)(A)(B) \rangle$ et $\langle (B)(A)(3) \rangle$.

Suite contenue dans une autre avec une fenêtre d'identité temporelle

Deux événements sont considérés comme simultanés s'ils arrivent dans une fenêtre de temps donnée, c'est-à-dire que leur distance temporelle est inférieure à un laps de temps fixé d'avance. La définition est alors presque la même que la précédente, avec une nouvelle notion d'inclusion.

Définition 15.11

On dira que la suite $S_1 = \langle a_1, \dots, a_n \rangle$ est incluse dans la suite $S_2 = \langle b_1, \dots, b_m \rangle$, $m \geq n$, si il existe une suite d'entiers $i_1 \leq u_1 \leq i_2 \leq u_2 \dots \leq i_n \leq u_n$ telle que pour chaque paire $b_{u_j} b_{i_j}$ telle que $\text{temps}(b_{u_j} - b_{i_j}) \leq \text{fenêtre}$, a_j est incluse dans l'union des b_k pour k compris entre $i - j$ et u_j .

Intuitivement, cela signifie que l'on transforme la suite des enregistrements en ajoutant les enregistrements obtenus en réunissant en un seul tous ceux qui arrivent dans la fenêtre temporelle.

Exemple

Considérons les deux itemsets précédents auxquels on rajoute la connaissance temporelle en indice (en jours) comme suit (cela revient à conserver les « data-sequences ») :

$$\begin{aligned} e_1 &= \langle (1)_{t=1}, (2)_{t=2}, (3, 4)_{t=15} \rangle \\ e_2 &= \langle (1, 2)_{t=1}, (5)_{t=20}, (3)_{t=50} \rangle \end{aligned}$$

Supposons que nous mettions une fenêtre temporelle de sept jours. Alors les données deviennent :

$$e_1 = \langle (1), (2), (1, 2), (3, 4) \rangle$$

$$e_2 = \langle (1, 2), (5), (3) \rangle$$

On obtient donc les suites fréquentes :

$$\langle (1)(3) \rangle, \langle (2)(3) \rangle \text{ et } \langle (1, 2)(3) \rangle.$$

Suite contenue dans une autre en présence d'un intervalle de validité

Deux événements ne seront ici considérés que s'ils ne sont pas trop éloignés temporellement, c'est-à-dire si leur distance temporelle est inférieure à une valeur *max-interv* fixée. La définition est alors la même que 15.7, moyennant encore une nouvelle notion d'inclusion :

Définition 15.12

On dira que la suite $S_1 = \langle a_1, \dots, a_n \rangle$ est incluse dans la suite $S_2 = \langle b_1, \dots, b_m \rangle$, $m \leq n$, si il existe une suite d'entiers $i_1 < i_2 \dots < i_n$ telle que $a_1 \subseteq b_{i_1}, a_2 \subseteq b_{i_2}, \dots, a_n \subseteq b_{i_n}$ où on ajoute la condition suivante : $\forall k > 1, \text{temps}(b_{i_k} - b_{i_{k-1}}) \leq \text{max-interv}$.

Intuitivement, *max-interv* élimine de l'appartenance à une séquence les événements qui sont trop lointains les uns des autres.

En ajoutant la condition $\forall k > 1, \text{temps}(b_{i_k} - b_{i_{k-1}}) \leq \text{min-interv}$, cette définition s'étend au cas où on désirerait éliminer aussi des événements trop rapprochés.

Exemples:

Considérons $e_1 = \langle (1)_{t=1}, (2)_{t=1}, (3, 4)_{t=1} \rangle$ $e_2 = \langle (1, 2)_{t=1}, (5)_{t=1}, (3)_{t=1} \rangle$ avec un *max-interv* de 20, alors plus aucune suite n'est fréquente car elles ne sont plus incluses dans e_2 .

Avec ce même *max-interv*, et en introduisant les relations de parentés ci-dessus : $e_1 = \langle (1, B)_{t=1}, (2, A)_{t=2}, t, (3, B, 4, A)_{t=15} \rangle$ $e_2 = \langle (1, B, 2, A)_{t=1}, (5, A)_{t=20} \rangle$ on observe donc les séquences fréquentes au sein de *max-interv* :

$\langle (B)(A) \rangle, \langle (A)(A) \rangle$

ainsi que leurs instances $\langle (1)(A) \rangle, \langle (2)(A) \rangle$.

Combinaison des cas précédents

On peut combiner les cas précédents en remplaçant \subseteq par « contenu dans » et la condition $\forall k > 1, \text{temps}(b_{i_k} - b_{i_{k-1}}) \leq \text{max-interv}$ par celle-ci : $\forall k > 1, \text{temps}(b_{u_k} - b_{i_{k-1}}) \leq \text{max-interv}$.

15.5.3.4 Génération de k -suites fréquentes à partir de $(k-1)$ -suites fréquentes

Définition: suites contiguës

Considérons l'exemple de la suite $S_2 = \langle (1, 2), (3, 4), (5), (6) \rangle$. La suite $\langle a_1, \dots, a_n \rangle$ est contiguë à la suite $S_2 = \langle b_1, \dots, b_m \rangle$, $n \leq m$ si au moins l'une des trois conditions suivantes est vérifiée :

1. S_1 est dérivée de S_2 en enlevant un item soit de b_1 , soit de b_m .

Par exemple, sont dérivées par cette règle de $S_2 = \langle (1, 2), (3, 4), (5), (6) \rangle$ les suites :

$\langle (1), (3, 4), (5), (6) \rangle, \langle (2), (3, 4), (5), (6) \rangle$ et $\langle (1, 2), (3, 4), (5) \rangle$.

2. S_1 est dérivée de S_2 en enlevant un item d'un quelconque b_i , à condition que b_i contienne au moins deux items.

Par exemple, sont dérivées par cette règle de $S_2 = \langle (1, 2), (3, 4), (5), (6) \rangle$ les suites :

$\langle (1), (3, 4), (5), (6) \rangle, \langle (2), (3, 4), (5), (6) \rangle, \langle (1, 2), (3), (5), (6) \rangle, \langle (1, 2), (4), (5), (6) \rangle$

En combinant les deux, on peut dériver les suites $\langle (1), (3), (5), (6) \rangle, \langle (1), (4), (5), (6) \rangle, \langle (2), (3), (5), (6) \rangle, \langle (2), (4), (5), (6) \rangle$, etc.

3. S_1 est contiguë à S'_1 et S'_1 est contiguë à S_2 . Par exemple, $\langle (3, 4), (5), (6) \rangle$ est contiguë à $\langle (1), (3, 4), (5), (6) \rangle$, laquelle est contiguë à $\langle (1, 2), (3, 4), (5), (6) \rangle$. En combinant à nouveau sur $\langle (1), (3, 4), (5) \rangle$, on obtient $\langle (4), (5) \rangle$ qui est donc contiguë à $\langle (1, 2), (3, 4), (5), (6) \rangle$.

Inversement, $\langle (1, 2), (3, 4), (6) \rangle$ ne peut être obtenue qu'en éliminant un élément ne contenant qu'un item non placé en bout de chaîne, donc elle n'est pas contiguë à $\langle (1, 2), (3, 4), (5), (6) \rangle$.

Intuitivement, on obtient l'ensemble des sous-suites contiguës d'une suite S en la « vidant » de la façon suivante :

- soit en enlevant un item aux éléments de S qui contiennent plus d'un item,
- soit en enlevant le premier ou le dernier élément s'ils ne contiennent qu'un seul item.

On va donc éliminer les éléments non centraux les uns après les autres. Il y a bien sûr de nombreuses façons d'effectuer cette opération.

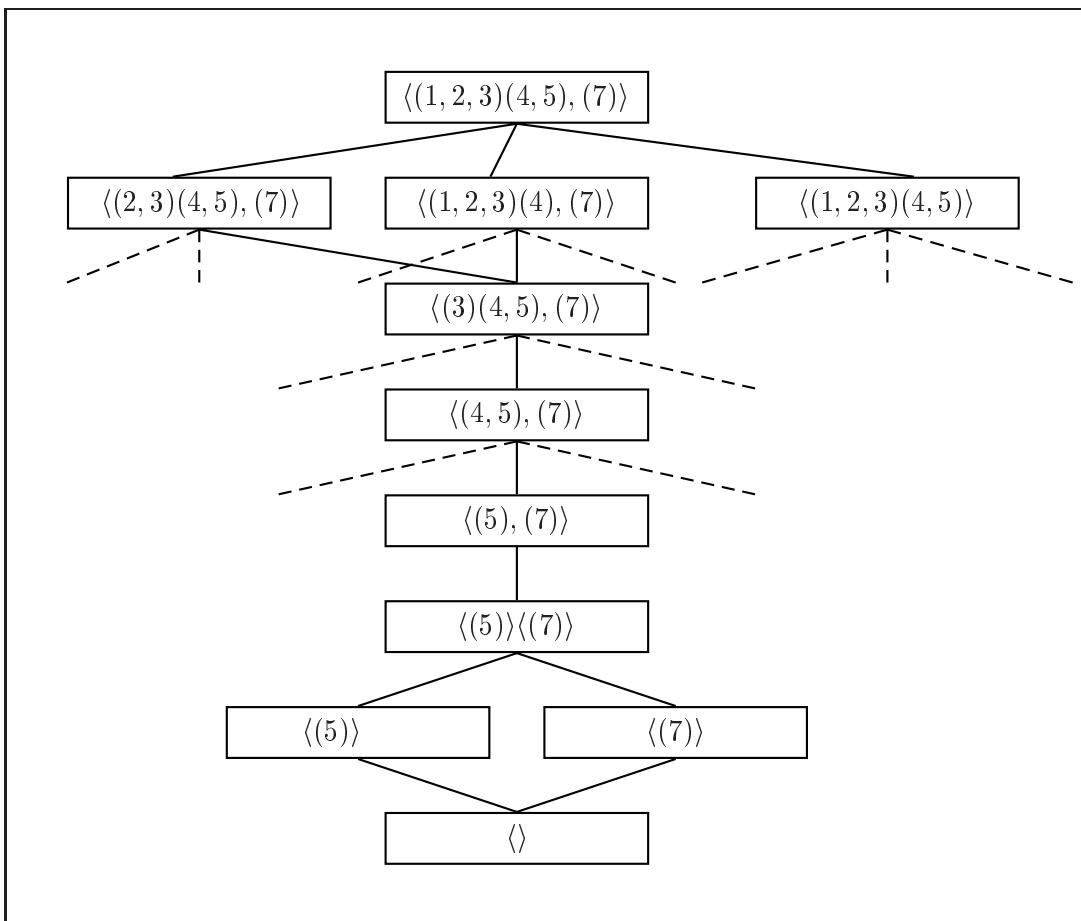
Exemple

Jointure de deux suites

Les suites S_1 et S_2 peuvent se joindre quand la suite obtenue en éliminant le premier élément de S_1 est la même que la suite obtenue en éliminant le dernier élément de S_2 .

Par exemple, $\langle (1, 2), (3, 4), (5) \rangle$ et $\langle (2), (3, 4), (5, 6) \rangle$ peuvent se joindre. Leur jointure est $\langle (1, 2), (3, 4), (5, 6) \rangle$.

Noter que la jointure de $\langle (1, 2), (3, 4) \rangle$ et $\langle (2), (3, 4), (5) \rangle$ est $\langle (1, 2), (3, 4), (5) \rangle$.



Dans le cas des 1-suites, on engendre ainsi deux jointures. Par exemple, en joignant les 1-suites $\langle(1)\rangle$ et $\langle(2)\rangle$, on doit introduire les 2-suites $\langle(1), (2)\rangle$ et $\langle(1, 2)\rangle$ qui toutes les deux redonnent une des listes de départ en enlevant soit le premier soit le deuxième élément.

On notera que les $k - 1$ -suites de départ sont contiguës aux k -suites d'arrivée.

On élimine alors des suites ainsi obtenues celles qui ont au moins une sous-suite contiguë qui n'a pas la couverture minimale.

Théorème 15.1

Cette procédure (à partir des $k - 1$ -suites respectant la contrainte sur max-interv) engendre un ensemble de k -suites qui contient l'ensemble de toutes les k -suites fréquentes (respectant la contrainte sur max-interv).

Ce théorème se démontre en passant par le lemme suivant :

Théorème 15.2

Soit D une suite indicée par le temps qui contient une sous-suite S (non indexée par le temps mais satisfaisant la contrainte max-interv). Toutes les sous-suites contiguës de S satisfont cette même contrainte.

En effet, si S satisfait la contrainte, cela signifie que la distance temporelle entre chacun de ces éléments est inférieure à max-interv . Alors, si on enlève un élément central à S , on peut introduire un nouvel intervalle de temps supérieur à max-interv .

Exemple sur une suite indicée.

La suite $\langle (1)_{t=1}, (2)_{t=10}, (3, 4)_{t=15} \rangle$ satisfait la contrainte $\text{max-interv} = 10$, alors que la sous-suite $\langle (1)_{t=1}, (3, 4)_{t=15} \rangle$ ne la satisfait pas. Cette opération d'enlever un élément central est justement interdite quand on engendre des sous-suites contigües. Quand on enlève un élément en tête ou en queue de suite, on supprime un intervalle de temps sans modifier les autres.

La suite $\langle (1)_{t=1}, (2)_{t=10}, (3, 4)_{t=15} \rangle$ satisfait la contrainte $\text{max-interv} = 10$ et les sous-suites $\langle (2)_{t=10}, (3, 4)_{t=10} \rangle$ et $\langle (1)_{t=1}, (2)_{t=10}, (3)_{t=15} \rangle$ la satisfont aussi, ainsi que toutes les sous-suites contigües: $\langle (2)_{t=10}, (3)_{t=15} \rangle$, $\langle (2)_{t=10}, (4)_{t=15} \rangle$, $\langle (1)_{t=1}, (2)_{t=10} \rangle$.

Les sous-suites contenant un seul élément satisfont trivialement la contrainte sur max-interv . Le théorème devient alors facile à démontrer. En effet, la phase de jointure ne peut engendrer que des séquences respectant la contrainte max-interv .

Exemple

Quel que soit max-interv , si les suites $\langle (1)_{t=1}, (2)_{t=10}, (3, 4)_{t=15} \rangle$ et $\langle (2)_{t=10}, (3, 4)_{t=15}, (6)_{t=25} \rangle$ satisfont à la contrainte, alors la suite jointe $\langle (1)_{t=1}, (2)_{t=10}, (3, 4)_{t=15}, (6)_{t=25} \rangle$ satisfait trivialement la contrainte. Du fait que la phase d'élimination ne crée que des sous-suites contigües, le lemme montre que toutes les sous-suites ainsi créées respectent la contrainte.

15.6 Le coapprentissage et les mélanges d'exemples supervisés et non supervisés

Nous présentons dans ce paragraphe deux techniques récentes qui permettent, sous certaines hypothèses, de superviser totalement un ensemble d'apprentissage composé d'une partie supervisée et d'une partie non supervisée. C'est particulièrement utile dans le cas où l'étiquetage par expert, qui coûte cher, doit être fait sur des grandes quantités de données, comme pour le traitement de la langue écrite ou orale.

15.6.1 Le cas de deux jeux indépendants d'attributs : le coapprentissage

Une première technique peut s'appliquer quand on dispose sur les données d'assez d'attributs pour les partager en deux sous-ensembles statistiquement indépendants. Blum et Mitchell ([BM98]) proposent l'exemple de la classification des pages Web des sites universitaires en classes telles que *pages personnelles des étudiants*, *page d'accueil d'un département*, *page de description d'un cours*, etc. Les deux jeux de descripteurs sont les suivants :

- Un vecteur entier de dimension égale à la taille du dictionnaire utilisé (le nombre de mots différents possibles), dont la valeur d'une coordonnée est le nombre de fois que ce mot apparaît dans la page Web. Cette description en « sac de mots » est très utilisée en linguistique automatique et donne en général des résultats plutôt bons¹¹ si on considère qu'elle ne tient pas compte de l'ordre des mots.
- Un autre vecteur sac de mots qui ignore le texte de la page et décrit ses références à d'autres pages (ses hyper liens).

11. Pour la tâche d'apprentissage du concept « page de description d'un cours », un classificateur bayésien naïf a été entraîné sur douze pages Web avec une description en sac de mots. Il a une performance de l'ordre de 87 % de bonne classification sur un ensemble de test de deux cent cinquante pages.

Ces deux descriptions sont indépendantes et pourtant les mots qui composent les hyperliens d'une page donnée sont d'une certaine façon une bonne description de cette page¹².

Notons d'une manière générale ces deux ensembles d'attributs indépendants \mathcal{X}_1 et \mathcal{X}_2 et notons \mathcal{S}_{sup} la partie supervisée de l'ensemble d'apprentissage et \mathcal{S}_{nonsup} la partie non supervisée. La technique dite de *coapprentissage (co-training)* se déroule comme indiqué dans l'algorithme 15.4. Elle y est décrite pour l'apprentissage d'un concept (deux classes), mais son extension est immédiate. Le cœur de la méthode repose sur le choix des exemples en nombre $p_1 + n_1 + p_2 + n_2$

Algorithme 15.4 Algorithme de coapprentissage

tant que la convergence n'est pas réalisée faire

Apprendre un classificateur A sur \mathcal{S}_{sup}

Apprendre un classificateur B sur \mathcal{S}_{sup}

Classer \mathcal{S}_{nonsup} par A

Classer \mathcal{S}_{nonsup} par B

Choisir les p_1 exemples positifs et n_1 exemples négatifs de \mathcal{S}_{nonsup} les plus sûrs pour A

Choisir par B p_2 exemples positifs et n_2 exemples négatifs de \mathcal{S}_{nonsup} les plus sûrs pour B

Ajouter ces $p_1 + n_1 + p_2 + n_2$ nouvellement classés à \mathcal{S}_{sup}

fin tant que

que l'on rajoute à chaque étape à \mathcal{S}_{sup} . Il faut pour les choisir classer par A et B tout l'ensemble \mathcal{S}_{nonsup} . On retient alors les p_1 exemples pour lesquels A est « le plus sûr » qu'ils sont positifs. De même pour les n_1 négatifs : ce sont ceux pour lesquels la décision de A est la plus sûre. C'est également ainsi que p_2 et n_2 autres exemples sont sélectionnés par B . Un algorithme de classification peut en effet avoir une mesure naturelle de « sûreté » : par exemple une probabilité, dans le cas d'une classification bayésienne, une distance à l'hyperplan appris dans le cas d'une décision linéaire, etc.

La justification empirique de cette méthode peut se faire ainsi : si le classificateur A trouve dans les données non supervisées un exemple très proche d'un des ses exemples d'apprentissage, il a de bonnes chances de le classer correctement. Mais cela ne signifie en rien que le classificateur B l'aurait classé correctement, puisque les jeux d'attributs \mathcal{X}_1 et \mathcal{X}_2 sont indépendants : être proches dans le premier espace n'implique pas que l'on soit proches dans le second. Par conséquent, A a ajouté à \mathcal{S}_{sup} un exemple supervisé qui va apporter de l'information à B .

Les expériences montrent que cette technique est efficace¹³. Son analyse théorique dans le cadre *PAC* prouve sa convergence sous certaines conditions. Cette méthode possède aussi des liens statistiques avec les méthodes de rééchantillonnage (voir le chapitre 11). En pratique, il est important de noter que l'indépendance effective des deux jeux d'attributs est déterminante pour son succès.

15.6.2 L'utilisation de l'algorithme EM

D'une manière générale, comment utiliser la partie supervisée \mathcal{S}_{sup} des exemples pour étiqueter \mathcal{S}_{nonsup} , la partie non supervisée ? Nous allons décrire une procédure fondée sur l'algorithme EM ,

12. Les résultats expérimentaux montrent que la classification obtenue sur ces seuls attributs sont presque équivalents aux précédents sur la même tâche

13. Le même concept appris sur un ensemble de deux cent cinquante pages étiqueté par co-apprentissage (dont douze étaient étiquetés au début) classe un ensemble de test indépendant en moyenne avec 95 % de succès. Les attributs sont maintenant l'union des ensembles \mathcal{X}_1 et \mathcal{X}_2 ([BM98]).

qui a été instanciée sur un certain nombre d'applications diverses. Rappelons que l'algorithme *EM*, dont l'annexe 18.9 donne une description, a déjà été présenté comme utile pour l'apprentissage des HMM (chapitre 13) et pour l'estimation des paramètres des mélanges de distribution gaussiennes, au paragraphe 15.2.2 de ce même chapitre.

Pour simplifier, plaçons-nous dans le cas de deux classes, ce qui n'est en aucune façon limitatif et donnons une version informelle du déroulement de cette méthode.

Il faut d'abord disposer d'un algorithme d'apprentissage fondé sur l'estimation de paramètres d'un modèle statistique. Typiquement, cet algorithme suppose par exemple que la distribution *a priori* de chacune des classes est gaussienne : les paramètres à estimer sont alors la moyenne et la matrice de covariance de la distribution de chaque classe. Ce problème a été traité dans le cas supervisé au chapitre 14. Par conséquent, une fois l'apprentissage par estimation des paramètres réalisé sur les données étiquetées \mathcal{S}_{sup} , on peut calculer pour chacune des deux classes la probabilité estimée d'avoir engendré chaque exemple de \mathcal{S}_{sup} et de \mathcal{S}_{nonsup} .

Il est possible maintenant d'étiqueter chaque exemple par la règle *MAP*, autrement dit en utilisant le principe de la classification bayésienne. Chaque exemple se verra donc attribuer l'étiquette de la classe qui a la plus grande probabilité de l'avoir engendré.

On peut désormais apprendre deux nouveaux modèles sur l'ensemble des données, en utilisant l'étiquetage que l'on vient de réaliser. Ces deux nouveaux modèles permettront un nouvel étiquetage de l'ensemble des données par le principe *MAP*, ce qui permettra de calculer deux modèles modifiés, et ainsi de suite. Cette boucle sera poursuivie jusqu'à ce que l'étiquette d'aucun exemple ne change plus.

D'une manière plus générale, on peut considérer la classe de chaque exemple comme une variable cachée et utiliser l'algorithme *EM* pour l'estimer, avec pour résultat l'étiquetage de chaque exemple par une classe. Ceci conduit à l'algorithme général 15.5.

Cette méthode générale a été particularisée pour l'apprentissage dans des problèmes variés où l'étiquetage est coûteux, comme en traitement de la langue naturelle écrite ou orale. On a ainsi réalisé l'apprentissage de modèles statistiques de séquences pour la syntaxe et le vocabulaire en reconnaissance de la parole, celui de modèles gaussiens naïfs pour la classification des sites Web, etc. Dans le cas où les modèles statistiques utilisés sont les modèles bayésiens naïfs, l'algorithme se réécrit sous une forme qui permet d'y ajouter des sophistications ([BM98]).

Algorithme 15.5 Algorithme supervisé non supervisé

Apprendre les modèles initiaux des classes sur \mathcal{S}_{sup}

$\mathcal{S} \leftarrow \mathcal{S}_{sup} \cup \mathcal{S}_{nonsup}$

tant que la convergence des modèles des classes n'est pas réalisée **faire**

Etape E (estimation) : étiqueter \mathcal{S} par le principe *MAP*

 selon les modèles courant des classes

Etape M (maximisation) : estimer les modèles des classes à partir de cet étiquetage

fin tant que

Notes historiques et sources bibliographiques

Les sources philosophiques de la classification sont anciennes et nombreuses et la classification automatique est toujours l'objet de débats de fond. L'argument de D. Hume, relayé par J.-L. Borges, est en effet simple et confondant : « Il n'existe pas de classification de l'univers qui ne soit

arbitraire et conjecturale. La raison en est fort simple: nous ne savons pas ce qu'est l'univers ». Sur un plan opérationnel, s'il ne s'agit que d'opposer les classes par leur nature géométrique et statistique (et non pas de découvrir leur nature cachée), les travaux remontent à Pearson (1894). Les mesures statistiques et les algorithmes ont été développés en particulier dans le cadre des sciences naturelles, mais aussi en reconnaissance de formes et en statistique appliquée. Les besoins actuels de la fouille de données ont donné une nouvelle impulsion à ce domaine, en particulier par l'introduction des techniques de découverte des associations entre attributs binaires, l'étude des mélanges supervisés et non supervisés et l'invention du co apprentissage. L'ouvrage de Jain et Dubes [JD88] est une somme théorique et pratique constamment citée en classification automatique « classique ». Mais les ouvrages en français sont nombreux et reflètent la vigueur de ce domaine en France: [Cel89], [Ler81], [Sap90], [Jam89], [Leb95].

Résumé

- Il existe des méthodes pour séparer en classes un ensemble d'apprentissage non supervisé.
- Ces méthodes peuvent induire une hiérarchie de partitions sur l'ensemble ou une partition avec un nombre donné de classes.
- Ces méthodes s'appliquent naturellement aux données numériques, mais peuvent s'étendre aux données binaires ou symboliques.
- D'autres techniques permettent d'extraire des associations logiques entre les attributs. Leur raffinement autorise la prise en compte d'intervalles temporels.
- Il est possible d'étiqueter complètement par co apprentissage des ensembles de données non complètement supervisés.

Chapitre 16

L'apprentissage de réflexes par renforcement

L'un des problèmes les plus fascinants en apprentissage est celui de l'adaptation en ligne, par renforcement. Il est en effet omniprésent dans les systèmes naturels, y compris chez les plus simples organismes, et correspond à une large classe d'applications dans laquelle il n'est pas envisageable de fournir les informations détaillées nécessaires à l'apprentissage supervisé. Dans sa forme la plus simple, cette situation d'apprentissage implique un système agissant dans le monde et soumis de ce fait à une séquence de signaux correspondant aux états successifs traversés par le système. De temps en temps, un signal de renforcement positif ou négatif sanctionne la séquence de décisions prises par le système. La tâche du système est de chercher une stratégie de conduite, appelée « politique » dans ce contexte, qui maximise l'espérance de renforcement dans les situations à venir. Cela passe généralement par une estimation des espérances de renforcement soit en fonction des états, soit en fonction des actions du système dans le monde.

L'apprentissage par renforcement est difficile pour deux raisons principales. D'une part, le signal de renforcement fourni en retour au système est très pauvre, c'est généralement juste un scalaire, et n'apporte donc que peu d'informations sur le monde et sur les décisions à prendre. D'autre part, le délai qui sépare le signal de renforcement des décisions qui y ont conduit rend ardue l'attribution de mérite ou de blâme à chacune des décisions prises dans le passé.

Malgré ces difficultés, l'apprentissage par renforcement a suscité de nombreux travaux depuis plus de quarante ans en automatique et en apprentissage artificiel. La notion d'un système autonome interagissant directement avec l'environnement et tirant de cette expérience une connaissance du monde suffisante pour y « survivre » et prospérer est en effet très séduisante intellectuellement. Par ailleurs, elle a de très nombreuses applications potentielles dans les domaines du contrôle de processus, de la navigation, de la conduite de robot, de l'apprentissage dans les jeux, de la planification financière, etc.

UN CANARD automatique, un programme, un robot peuvent-ils apprendre à se comporter dans un environnement inconnu, ou en tout cas dont ils ne perçoivent l'existence que par des réponses agréables ou désagréables à ses actions ? On sait qu'un chien se dresse par punition-récompense et que des animaux généralement jugés moins intelligents, comme les oies ou les corbeaux, sont capables de modifier durablement leur comportement en fonction d'essais et de réponses du monde extérieur. Mais ce type d'apprentissage, fondamental dans le monde animal, est-il modélisable et transposable pour des programmes ?

Il semble désormais que la réponse soit positive, grâce aux techniques de l'apprentissage par renforcement. Il est aujourd'hui possible d'imaginer un robot arrivant sur une planète inconnue où il doit tout découvrir : les éventuelles chausses-trappes, les endroits lui permettant de recharger ses batteries, l'action de ses roues dans cet environnement inconnu. Son objectif est de survivre le mieux possible dans ce monde. Pour cela, il doit identifier les différents *états* dans lesquels il peut se trouver, apprendre à associer des effets à ses actions pour chaque état possible et découvrir comment ce monde extérieur associe une réponse aux états.

Dans cette situation d'apprentissage, l'espace des hypothèses est particulier puisqu'il s'agit d'apprendre deux fonctions : l'une faisant passer d'état en état et l'autre faisant effectuer une action à partir d'un état, l'une et l'autre tenant compte de la récompense ou de la punition associée par le monde extérieur. Il s'agit donc d'un problème d'optimisation numérique, mais dans lequel les techniques par exploration définies au chapitre 3 ne sont que de peu d'utilité.

Naturellement, le robot de l'illustration ci-dessous n'est pas doté de facultés d'apprentissage ; il n'est pourtant pas impossible d'envisager des systèmes mécaniques améliorant leur comportement par apprentissage par renforcement, mais il est beaucoup plus facile d'utiliser des machines programmables pour cela.



FIG. 16.1 – Une réplique réalisée par Frédéric Vidoni en 1998 du canard de J. de Vaucanson (c. 1741). Musée des automates de Grenoble.

Notations utiles pour ce chapitre

\mathcal{E}	L'ensemble des états
\mathcal{Z}	L'ensemble des actions
\mathcal{R}	L'ensemble des signaux de renforcement (souvent \mathbb{R})
s	Un état
a	Une action
$\pi(s, a)$	La probabilité que l'action a soit choisie dans l'état s par la politique π
$Q^*(s, a)$	La vraie espérance de gain quand l'action a est prise dans l'état s
r_t	Le signal de renforcement reçu par l'agent à l'instant t
R_t	Le gain cumulé à partir de l'instant t
E_π	Espérance en suivant la politique π
$0 \leq \gamma \leq 1$	Le taux de diminution des renforcements

16.1 Description du problème

Nous nous intéressons à un agent situé dans un certain environnement qu'il ne connaît pas, ou du moins qu'il ne connaît qu'imparfaitement. Cet agent cherche à recevoir des récompenses et à éviter les punitions, qui dépendent de ses actions. Comment doit-il s'y prendre? Quelle ligne de conduite doit-il suivre?

Reprendons les choses plus formellement. Les récompenses et punitions, pour lesquelles nous utiliserons désormais le terme de *signal de renforcement*, dépendent de l'état présent du système. Plus précisément, nous supposons qu'à chaque état correspond un signal de renforcement, éventuellement nul. Si l'agent cherche à maximiser une certaine espérance de gain, il doit donc chercher à atteindre les états correspondant aux signaux les plus favorables (les récompenses). Pour cela, l'agent doit être capable de mesurer son état courant et d'identifier les actions qui sont les plus à même de le conduire vers les états favorables. Les problèmes à résoudre sont les suivants :

1. L'agent ne connaît pas son environnement, ce qui signifie :
 - qu'il ne connaît pas les signaux de renforcement associés à chaque état ;
 - qu'il ne connaît pas la topologie de l'espace des états, c'est-à-dire notamment les états accessibles à partir d'un état donné.
2. L'agent ne connaît pas l'effet de ses actions dans un état donné, c'est-à-dire la fonction qui associe à chaque couple (état, action) un état résultant.

Dans les méthodes d'apprentissage par renforcement, l'agent n'a pas de connaissances *a priori* sur le monde et il opère par une sorte de reconnaissance des états du monde et une interpolation de ses observations. Il est évident que ce type d'apprentissage par identification progressive nécessite une long apprentissage. En contrepartie de cette relative inefficacité, et parce qu'il s'accommode de présupposés très faibles, il s'agit d'un apprentissage applicable dans une grande variété de situations. En fait, la seule hypothèse est que le monde est de nature stochastique (les actions peuvent avoir des effets non déterministes) et *stationnaire* (les probabilités de transition entre états, et les signaux de renforcement, restent stables au cours du temps).

16.1.1 La modélisation d'un agent en action dans le monde

La théorie de l'apprentissage par renforcement s'appuie sur une modélisation des agents et des environnements qui est une idéalisation manipulable des situations réelles d'apprentissage,

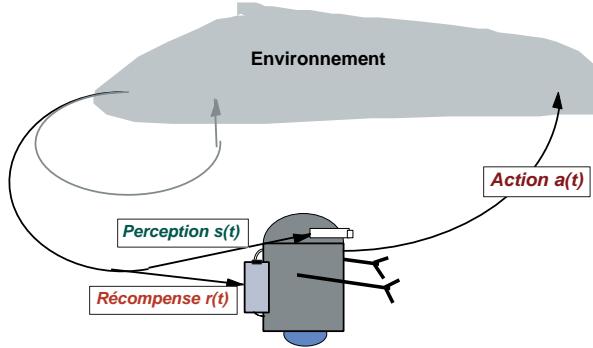


FIG. 16.2 – Le schéma abstrait d'un agent en interaction avec le monde suivant trois canaux : perception, renforcement immédiat et action instantanée.

sans être triviale au point de trop s'éloigner de la réalité. Cette modélisation suppose que l'agent communique avec son environnement par trois canaux distincts :

- Un *canal perceptif* par lequel l'agent mesure l'état dans lequel il se trouve dans l'environnement. Ce canal correspond aux données fournies par un ensemble de capteurs, par exemple des caméras, des capteurs de proximité à ultrasons, des centrales inertielles, etc. Les informations fournies sont souvent partielles et parfois erronées. Nous notons $s(t)$ l'ensemble des informations passant par ce canal à l'instant t .
- Un canal spécifique aux *signaux de renforcement* renseignant l'agent sur la qualité de l'état courant. On suppose dans l'apprentissage par renforcement que ce canal ne transmet qu'un scalaire¹. Nous notons $r(t)$ l'information transmise par ce canal. Ce signal n'est généralement pas disponible dans tous les états, mais seulement pour quelques états particuliers. Par exemple, c'est seulement à la fin d'une partie d'échecs que l'on dispose de la sanction : perte, gain ou nulle.
- Un canal qui transmet à l'environnement l'*action de l'agent*. Nous notons $a(t)$ l'information ainsi transmise de l'agent vers l'environnement. Généralement, ce signal déclenche une modification de l'état de l'environnement, comme par exemple quand un robot fait tourner ses roues ou quand un joueur d'échecs joue un coup. Ici aussi, la modification de l'état peut être non déterministe dans la mesure où l'agent n'a pas une connaissance parfaite de l'environnement.

Nous notons \mathcal{E} l'espace des états mesurables, \mathcal{R} l'espace des signaux de renforcement, c'est-à-dire un intervalle de la forme $[-a, +b]$ avec $a, b \in \mathbb{R}^+$, et \mathcal{Z} l'espace des actions disponibles pour l'agent. Dans ce cadre, nous posons donc qu'à chaque instant t , l'agent perçoit le monde comme étant dans l'état $s_t \in \mathcal{E}$. Il choisit alors d'effectuer l'action $a_t \in \mathcal{Z}$ parmi les actions possibles dans l'état courant. À la suite de cette action prise dans cet état, il reçoit un signal de renforcement immédiat $r_t \in \mathcal{R}$.

L'agent peut donc être considéré comme réalisant une fonction de \mathcal{E} dans \mathcal{Z} : $s_t \mapsto a_t$. Suivant la terminologie en usage, nous appelons *politique* cette fonction de comportement, et nous notons π_t la politique à l'instant t . Plus précisément, une politique est une fonction définie de $\mathcal{E} \times \mathcal{Z}$ dans \mathbb{R} , qui associe à chaque état s et chaque action a possible dans s , la probabilité $\pi(s, a)$ associée de choisir l'action a dans s . Si les probabilités sont uniformément nulles sauf pour une action, l'agent est déterministe.

1. Dans le cas des organismes naturels, un certain précablage existe pour percevoir ce type de signal comme une douleur ou un plaisir plus ou moins forts.

L'environnement peut être vu pour sa part comme implémentant une fonction de $\mathcal{E} \times \mathcal{Z}$ dans $\mathcal{E} \times \mathcal{R}$: $(s_t, a_t) \mapsto (s_{t+1}, r_t)$. Pour des raisons de clarté, il est utile de décomposer cette fonction en deux fonctions. La première est une *fonction de transition entre états* notée T . Elle traduit la dynamique du monde; elle est définie de $\mathcal{E} \times \mathcal{A}$ dans \mathcal{E} : $(s_t, a_t) \mapsto s_{t+1}$. La seconde est une *fonction de renforcement immédiat* R de $\mathcal{E} \times \mathcal{Z}$ dans \mathcal{R} : $(s_t, a_t) \mapsto r_t$. Chacune de ces fonctions est stochastique, soumise à des aléas imprévisibles, qu'on suppose issus d'une distribution stationnaire.

16.1.2 Les notions fondamentales

Dans cette section, nous allons formaliser les concepts introduits par la théorie actuelle de l'apprentissage par renforcement et décrire les problèmes et les grandes familles d'approches.

L'apprentissage par renforcement considère un apprenant plongé dans un environnement et devant essayer, par ses actions, de maximiser une mesure de gain dépendant des signaux de renforcement reçus tout au long de son existence dans le monde. L'une des premières questions consiste donc à spécifier cette mesure de gain.

16.1.2.1 Les mesures de gain

Précisons d'emblée qu'il n'y a pas de mesure de gain universelle valable pour toutes les situations. Chaque domaine d'application est susceptible d'avoir sa mesure adaptée. Ainsi, le joueur d'échec est sans doute sensible au compte des pièces gagnées ou perdues en cours de partie, à certains critères tels que le contrôle du centre, mais ce qui l'intéresse avant tout est l'issue ultime de la partie: gain, perte ou nulle. En revanche, pour un fournisseur d'énergie électrique qui essaie de réguler sa production en fonction de la demande, laquelle dépend de la météorologie, de l'heure de la journée, de la situation économique, etc., il est important de mesurer les gains et les coûts tout au long du processus de production. La mesure de gain doit donc être différente dans ce cas. En général, on s'intéresse à une mesure de gain cumulée dans le temps. Ainsi, que l'on ne tienne compte, comme aux échecs, que du gain ultime, ou bien que l'on moyenne les gains réalisés en cours d'action, ou encore que l'on tienne compte, comme en économie, de gains pondérés par des taux d'intérêt, toutes les options sont envisageables et dépendent de l'application concernée. Cependant, trois mesures ont été plus particulièrement distinguées dans les recherches sur l'apprentissage par renforcement.

- *Gain cumulé avec horizon infini*:

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \dots + r_T = \sum_{i=t+1}^T r_i \quad (16.1)$$

- *Gain cumulé avec intérêt et horizon infini*:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots + r_T = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \quad (16.2)$$

où γ joue le rôle d'un taux d'intérêt : $0 \leq \gamma \leq 1$.

- *Gain en moyenne*:

$$R_t = \frac{1}{T-1} \sum_{i=t+1}^T r_i \quad (16.3)$$

Il faut noter que le choix de la mesure de gain a une grosse influence sur le choix de la meilleure politique par le système (voir la figure 16.3). Il est donc essentiel de peser soigneusement sa définition avant de se lancer dans le processus d'apprentissage.

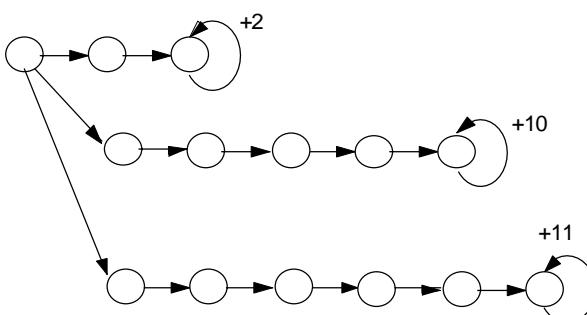
Avec $k = 4$ et $\gamma = 0.9$, quelle est la meilleure politique?	$\sum_{t=0}^k r_t$	$\sum_{t=0}^{\infty} \gamma^t r_t$	$\lim_{k \rightarrow \infty} \frac{1}{k} \sum_{t=0}^k r_t$
	6	16	2
	0	59	10
	0	58.4	11

FIG. 16.3 – Dans l'exemple ci-dessus, le choix de la meilleure stratégie, parmi les trois possibles, dépend du critère de gain adopté. Avec $k = 4$ et $\gamma = 0.9$, quelle est la meilleure politique dans l'état initial? Si l'on vise la première formule de gain, il faut choisir la première politique puisqu'elle conduit au meilleur gain. Pour la deuxième formule de gain, il faut choisir la seconde politique. Et pour la troisième formule, il faut choisir la troisième politique. D'après un exemple dû à [KLM96].

16.1.2.2 Le dilemme exploration contre exploitation

Plaçons-nous dans le cas d'un agent ne connaissant pas son environnement et cherchant à maximiser une mesure de gain cumulée. Pour ce faire, il doit naturellement commencer à explorer son univers pour en découvrir les potentialités, mais très vite il se trouve devant un dilemme. Ayant découvert que certains comportements semblent plus profitables que d'autres, doit-il chercher à les reproduire au maximum afin de maximiser le gain recherché au risque de passer à côté d'opportunités non encore identifiées, ou doit-il continuer son exploration au risque de perdre du temps et de ne pas réaliser un bon gain cumulé? Ce problème est classique dans tous les scénarios où un agent doit prendre des décisions «en ligne» et optimiser une mesure de performance le long de sa trajectoire. Une modélisation simple de ce genre de situations est celle d'un joueur placé devant l'une de ces machines que l'on appelle² le « bandit à deux bras » (voir figure 16.4).

Le principe est le suivant. Un joueur dispose de m jetons avec lesquels il peut jouer avec cette machine. Pour chaque jeton inséré dans la fente, le joueur peut tirer sur l'un ou l'autre des bras. Il reçoit alors un certain nombre de pièces correspondant à son gain. Les bras sont notés A_1 et A_2 , ils ont une espérance de gain respective de μ_1 et μ_2 avec une variance respective de σ_1^2 et σ_2^2 . Cela signifie que le gain associé à chaque bras est aléatoire, avec une certaine moyenne et un certain écart-type stationnaires. Les tirages aléatoires sont supposés indépendants. Le joueur ne connaît ni les moyennes ni les variances associées à chaque bras et doit donc les estimer en cours de jeu. Le joueur ne sait donc pas quel est le bras dont l'espérance de gain est la meilleure, et il doit essayer de maximiser son gain avec ses m jetons. Quelle doit alors être sa stratégie de tirage des bras étant donnée son estimation courante des moyenne et variance de chaque bras?

Une stratégie extrême est de tirer une fois chaque bras, de noter celui qui a donné le meilleur résultat, puis de jouer désormais systématiquement celui-ci. Cela correspond à une *stratégie*

2. La machine à sous classique de casino est appellée en argot américain « bandit manchot » (*one-armed bandit*).

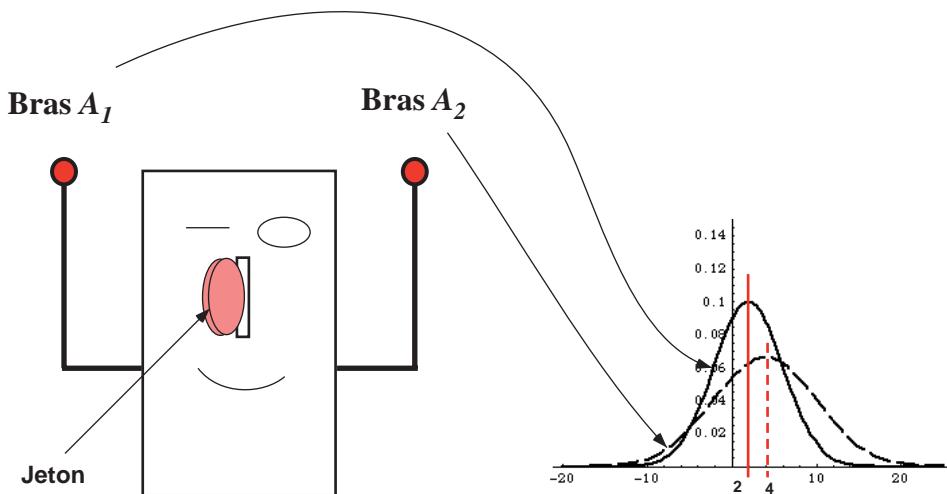


FIG. 16.4 – Une machine à sous appelée « bandit à deux bras ». Chacun des bras est associé à un gain aléatoire suivant une distribution normale. Par exemple, ici, le bras A₁ correspond à une loi de moyenne 4 et de variance 6, alors que le bras A₂ correspond à une loi de moyenne 2 et de variance 4. Il peut bien sûr arriver que sur un tirage le bras A₁ donne un résultat inférieur au résultat obtenu avec un tirage de A₂.

d'exploitation pure: ne plus explorer plus dès qu'on possède des éléments d'information minimaux. Bien sûr, le risque est que les deux tirages initiaux n'aient pas révélés le meilleur bras à cause de la variance des résultats et que le plus mauvais bras ait donc été tiré systématiquement. Plus généralement, on appelle stratégie d'exploitation toute stratégie qui choisit l'action dont l'estimation courante de gain est la plus élevée.

La stratégie extrême inverse consiste à tirer $\lfloor \frac{m-1}{2} \rfloor$ fois sur le bras gauche et $\lfloor \frac{m-1}{2} \rfloor$ fois sur le bras droit³, puis à tirer le dernier ou les deux derniers coups sur le bras dont la moyenne observée est la meilleure. Cela correspond à une *exploration pure*, dans laquelle on alloue quasiment toutes les décisions à l'exploration de l'environnement avant de choisir la décision ultime. Celle-ci est alors prise avec une connaissance aussi grande que possible, mais au prix de n'avoir pas cherché à optimiser le gain durant la phase d'exploration.

On sent bien que la stratégie optimale doit se situer entre ces deux types de politiques et qu'elle correspond à la résolution d'un compromis entre exploitation et exploration. Sa résolution analytique (voir par exemple [Hol75]) montre que, dans le cas du bandit à deux bras, la meilleure stratégie consiste, au fur et à mesure que de l'information est acquise sur les probabilités de gain de chaque bras, à accroître exponentiellement le nombre de tirages du bras paraissant le meilleur par rapport à l'autre. (On peut trouver également une analyse simplifiée du bandit-à-deux-bras dans [Mit96]).

Nous verrons dans la suite que la tâche d'apprentissage par renforcement implique également la résolution d'un conflit entre exploration et exploitation. Si les leçons générales du problème des bandits à deux bras restent valables, elles doivent être affinées pour chaque type d'environnement. En général, il est d'ailleurs impossible de d'obtenir une solution analytique par insuffisance de connaissances sur le modèle sous-jacent de l'environnement.

3. La notation $\lfloor x \rfloor$ indique l'entier immédiatement inférieur à x .

16.1.2.3 La mesure de performance de l'apprentissage

Dans ce qui précède, nous avons introduit les éléments de mesure des performances de l'agent dans son univers. Une autre question concerne la mesure de performance de l'apprentissage lui-même. Sans entrer dans les détails ici, il est clair qu'il s'agit d'un paramètre important. En effet, l'apprentissage par renforcement suppose un agent apprenant par essais et erreurs. Les erreurs peuvent coûter cher : soit parce qu'elles correspondent à des dégradations de l'environnement (par exemple si un robot heurte des objets ou un autre robot durant son apprentissage) ou à des dégradations de l'agent lui-même, ou encore parce qu'ils exigent de très longues sessions d'apprentissage. C'est pourquoi il est intéressant de caractériser l'apprentissage en fonction de ses propriétés de convergence vers une politique optimale, ou bien aussi en fonction de la complexité en calculs ou encore en nombre d'expériences (paires (*situation, action*)) nécessaires. Par exemple, un ordre de grandeur pour l'apprentissage du jeu d'Othello par renforcement est du million de parties d'essai à jouer par l'agent avant d'avoir un bon niveau de jeu. Ce qui est tolérable pour ce type de tâche, facile à simuler, peut l'être beaucoup moins lorsque l'apprentissage implique un agent réel, par exemple une sonde d'exploration planétaire.

On cherche donc autant que possible à analyser les propriétés de convergence asymptotique vers une politique optimale, ainsi que la complexité computationnelle et en nombre d'essais (mesurés en nombre d'actions ou de séquences d'actions par exemple) de cette convergence.

16.1.3 Les problèmes et les grandes approches

Le problème d'optimisation ayant été défini, plusieurs approches sont envisageables pour le résoudre.

La première consiste à chercher à *apprendre directement un modèle de l'environnement* en estimant d'une part la fonction de renforcement immédiat associée à chaque état, ou à chaque couple (état, action) R : définie de \mathcal{E} dans \mathcal{R} : $s_t \mapsto r_t$, et, d'autre part, la fonction de transition T caractérisant la dynamique de l'environnement : définie de $\mathcal{E} \times \mathcal{Z}$ dans \mathcal{E} : $(s_t, a_t) \mapsto s_{t+1}$. Le problème est alors celui d'un apprentissage supervisé s'appuyant sur les exemples glanés en cours d'expérience. L'inconvénient de cette approche, outre qu'elle nécessite de chercher à tester toutes les situations possibles, et cela plusieurs fois, est de ne pas tenir compte des interactions entre les états.

Une autre approche prend en compte ces interactions par l'introduction de *fonctions d'utilité*. Il s'agit de fonctions traduisant l'espérance de gain à partir d'un état : fonction notée $V(s)$ et définie sur \mathcal{S} , ou à partir d'un couple (état, action) : fonction notée $Q(s, a)$ et définie sur $\mathcal{E} \times \mathcal{Z}$. Ces fonctions estiment sur le long terme la qualité des états ou des couples (état, action). Elles sont donc à différencier des fonctions de renforcement immédiat. Dans ce cas, l'apprentissage consiste à agir dans le monde et à calculer pour chaque état ou couple (état, action) l'espérance de gain associée. Évidemment, ce type d'apprentissage, *a priori* plus intéressant puisqu'il permet un choix local de l'action à prendre, celle qui maximise l'utilité, introduit aussi des contraintes particulières.

Finalement, il est possible d'envisager de *travailler directement dans l'espace des politiques* plutôt que de passer par l'intermédiaire de fonctions locales aux états. C'est par exemple ainsi qu'opère le mécanisme de sélection darwinienne dans la nature (voir le chapitre 8). Chaque agent correspond à une certaine politique, et en moyenne les agents de performances médiocres sont éliminés au profit d'agents supérieurs. Ce type d'approche a également fait l'objet d'expériences dans le cadre de modèles d'évolution simulée (voir chapitre 8). Le succès dépend en grande partie de la structuration de l'espace des politiques. Si celle-ci est faible, il faut avoir recours aux méthodes d'apprentissage de fonctions locales, c'est-à-dire aux méthodes d'apprentissage

par renforcement proprement dites que nous allons désormais examiner.

16.2 Si tout est connu : l'utilité de la fonction d'utilité

L'idée profonde sous-jacente à l'apprentissage par renforcement est de permettre à l'agent d'optimiser sa conduite sur le long terme (celle qui maximise son gain cumulé) sur la base de décisions locales ne nécessitant pas de recherche en avant. Il faut donc que l'information disponible localement, au moment et au lieu où se trouve l'agent, reflète l'espérance de gain à long terme. Cette information locale, résumant les potentialités à long terme des actions possibles, est traduite par une valeur numérique appelée *utilité*. Plus généralement, on parle de *fonction d'utilité* pour désigner la fonction associant à chaque état, ou à chaque paire (*état, action*), sa valeur d'utilité. Nous étudierons deux fonctions d'utilité particulières : la première associe à chaque état s l'espérance de gain à partir de cet état si l'on suit la politique π :

$$V^\pi(s) = E_\pi\{R_t|s_t = s\} \quad (16.4)$$

la seconde associe à chaque couple (*état, action*) (s, a) l'espérance de gain en partant de l'état s , en effectuant l'action a , puis en suivant la politique π :

$$Q^\pi(s, a) = E_\pi\{R_t|s_t = s, a_t = a\} \quad (16.5)$$

Dans le cas d'un gain cumulé avec intérêt et horizon infini, les deux formules deviennent :

$$\begin{aligned} V^\pi(s) &= E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right\} \\ Q^\pi(s, a) &= E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\right\} \end{aligned}$$

Nous venons de définir l'espérance de gain, donc l'utilité, en fonction d'une politique donnée. Il se trouve que l'on peut définir une relation d'ordre partiel sur les politiques en fonction des valeurs d'utilité associées⁴. Plus précisément :

Définition 16.1 (Ordre sur les politiques)

Une politique π est dite supérieure à une autre politique π' si et seulement si l'espérance de gain suivant π est supérieure ou égale à l'espérance de gain suivant π' pour tous les états $s \in \mathcal{E}$. En d'autres termes, $\pi \geq \pi'$ ssi $V^\pi(s) \geq V^{\pi'}(s), \forall s \in \mathcal{E}$.

Cet ordre partiel permet de définir le concept de *politique optimale* :

Définition 16.2 (Politique optimale π^*)

Une politique optimale est une politique supérieure ou égale à toutes les autres politiques. Dans le cas des processus markoviens, il en existe toujours une. On note π^ cette politique optimale, ou les politiques optimales s'il y en a plusieurs.*

4. Plus précisément, cette relation d'ordre existe dans le cas des *processus markoviens*, pour lesquels on suppose que la donnée d'un état suffit à résumer toute l'histoire antérieure du système (voir le chapitre 13). En d'autres termes, les décisions peuvent alors être prises sans connaître l'histoire passée du système. Cette hypothèse est toujours faite dans la théorie de l'apprentissage par renforcement.

On a alors les fonctions d'utilité correspondantes :

$$V^*(s) = \max_{\pi} V^\pi(s), \quad \forall s \in \mathcal{E} \quad (16.6)$$

$$Q^*(s, a) = \max_{\pi} Q^\pi(s, a), \quad \forall s \in \mathcal{E} \text{ et } \forall a \in \mathcal{Z} \quad (16.7)$$

En les utilisant, il devient facile d'obtenir un comportement optimal.

Supposons que l'agent dispose des valeurs $V^*(s), \forall s \in \mathcal{S}$. Placé dans l'état s_t , l'agent n'a qu'à faire une recherche en avant à un pas pour déterminer l'action à prendre. Pour chaque action a disponible, il examine les états accessibles s' (il peut y en avoir plusieurs pour une même action si l'environnement est non déterministe, chacun avec une certaine probabilité d'obtention) et note pour chacun d'eux sa valeur $V^*(s')$. L'action a^* associée avec la meilleure espérance de gain est l'action qu'il faut choisir :

$$a_t^* = \operatorname{ArgMax}_{a \in \mathcal{Z}} \sum_{s'} \mathcal{P}_{s_t s'}^a \left[\mathcal{R}_{s_t s'}^a + \gamma V^*(s') \right] \quad (16.8)$$

Supposons maintenant que l'agent dispose des valeurs $Q^*(s, a), \forall s \in \mathcal{E}$ et $\forall a \in \mathcal{Z}$. Dans ce cas, la détermination de la conduite optimale dans la situation courante s_t est encore plus simple, il suffit de choisir l'action a maximisant $Q^*(s, a)$:

$$a_t^* = \operatorname{ArgMax}_{a \in \mathcal{Z}} Q^*(s_t, a) \quad (16.9)$$

Dans les deux cas, la décision peut donc être obtenue très facilement. Si l'on dispose des valeurs d'utilité $V^*(s)$, il faut faire une recherche en avant d'un pas, tandis que cela n'est même pas nécessaire dans le cas où l'on dispose des valeurs d'utilité $Q^*(s, a)$. Toutefois, la différence entre les deux méthodes est plus profonde qu'il n'y paraît. L'équation (16.8) nécessite en effet la connaissance de l'environnement sous la forme des probabilités de transition $\mathcal{P}_{ss'}^a$, et des renforcements associés $\mathcal{R}_{ss'}^a$ reçus lorsque l'agent passe de l'état s à l'état s' sous l'action a . Cette connaissance n'est pas nécessaire dans le cas de l'équation (16.9). Bien sûr, cette énorme simplification se paie par le fait qu'il faut maintenant travailler dans l'espace produit $\mathcal{E} \times \mathcal{Z}$ au lieu de l'espace \mathcal{E} .

16.3 L'apprentissage des fonctions d'utilité quand l'environnement est connu

Dans cette section, nous allons supposer que l'agent a une connaissance de l'environnement sous la forme des probabilités de transition $\mathcal{P}_{ss'}^a$ et des renforcements associés $\mathcal{R}_{ss'}^a$. En revanche, il ne connaît pas les fonctions d'utilité. Cela signifie qu'il a une connaissance locale de son environnement, mais qu'il n'a pas d'information sur l'impact à long terme des décisions qu'il pourrait prendre. Comment peut-il alors les apprendre ?

Nous allons avoir deux problèmes à résoudre. D'une part, nous l'avons vu, une fonction d'utilité dépend de la politique suivie. Il nous faut donc voir comment apprendre ces fonctions dans le cadre d'une politique donnée. D'autre part, nous cherchons à obtenir, ou du moins à approcher, une politique optimale. Il nous faut donc voir comment passer de fonctions d'utilité associées à des politiques *a priori* sous-optimales à la détermination d'une politique optimale. Nous allons étudier ces deux problèmes l'un après l'autre.

16.3.1 L'évaluation d'une politique par propagation locale d'information

Dans un premier temps, nous cherchons à évaluer une politique π en déterminant les valeurs $V^\pi(s), \forall s \in \mathcal{E}$.

Intuitivement, une approche simple serait de placer l'agent en chaque état s et de lui faire suivre à partir de là la politique π , éventuellement de nombreuses fois pour moyennner sur toutes les trajectoires possibles si le monde est non déterministe. Il suffirait alors de calculer la moyenne des gains cumulés obtenus pour avoir l'estimation de l'espérance de gains à partir de s . Nous allons développer la formule correspondante dans le cas du gain cumulé avec intérêt et horizon infini, mais ce qui importe est l'idée générale qui se transfère sans problème à d'autres formules de gains.

$$\begin{aligned}
V^\pi(s) &= E_\pi \left\{ R_t \mid s_t = s \right\} \\
&= E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right\} \\
&= E_\pi \left\{ r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_t = s \right\} \\
&= \sum_a \pi(s, a) \sum_{s'} \mathcal{P}_{ss'}^a \left[\mathcal{R}_{ss'}^a + \gamma E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_{t+1} = s' \right\} \right] \\
&= \sum_a \pi(s, a) \sum_{s'} \mathcal{P}_{ss'}^a \left[\mathcal{R}_{ss'}^a + \gamma V^\pi(s') \right]
\end{aligned} \tag{16.10}$$

Ce résultat est remarquable car il met en valeur le fait que l'on peut ramener un calcul de gain prenant en compte toute une trajectoire (ou un ensemble de trajectoires) à partir de s à un calcul de gain s'appuyant sur les estimations $V^\pi(s')$ des états s' accessibles à partir de l'état courant s . Il permet d'exploiter ainsi une dépendance ou corrélation entre les espérances de gain des états. L'utilité d'un état suivant la politique π est égale à une somme pondérée suivant les probabilités de transition aux états successeurs des utilités de ces états successeurs plus le signal de renforcement reçu lors de la transition de s à s' . C'est l'*équation de Bellman*⁵ pour V^π .

Il se trouve que l'on peut démontrer que la valeur $V^\pi(s)$ est l'unique solution de l'équation de Bellman. Par ailleurs, l'équation de Bellman peut être utilisée dans une procédure d'approximation itérative :

$$\begin{aligned}
V_{k+1}(s) &= E_\pi \{ r_{t+1} + \gamma V_k(s_{t+1}) \mid s_t = s \} \\
&= \sum_a \pi(s, a) \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V_k(s')]
\end{aligned} \tag{16.11}$$

pour tout $s \in \mathcal{E}$. Il est clair que $V_k = V^\pi$ est un point fixe de cette règle itérative. Par ailleurs, il est possible de montrer que la séquence $\{V_k\}$ converge vers V^π lorsque $k \rightarrow \infty$ si $\gamma < 1$ ou si les gains sont calculés sur un horizon limité. Intuitivement, cette convergence se justifie par le fait que chaque mise à jour d'une valeur $V_k(s)$ s'appuie sur d'autres estimations $V_k(s')$, mais aussi sur le signal de renforcement observé r_{t+1} . Il y a donc bien un gain d'information sur l'environnement à chaque itération.

5. Cette équation fait intervenir le principe de Bellman, qui est à la base des méthodes de programmation dynamique. Ce principe a été appliqué par exemple au chapitre 13 dans l'algorithme de Viterbi.

Algorithme 16.1 Algorithme d'évaluation itérative d'une politique.

Donnée: la politique π à évaluer

Initialisation: $V(s) = 0$, pour tous $s \in \mathcal{E}^+$ (les états accessibles depuis s)

$\Delta \leftarrow 0$

Pour tout $s \in \mathcal{E}$:

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_a \pi(s, a) \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

$\Delta < \theta \{(\text{un petit nombre réel positif})\}$

Sortie $V \approx V^\pi$

16.3.2 Un théorème conduisant à l'amélioration de politique

Nous avons vu comment il était possible en principe d'approcher l'espérance de gain en chaque état pour une politique donnée. Cela permet à l'agent de prendre des décisions rapides sur la base d'informations locales. Cependant, l'agent cherche aussi à améliorer sa politique, et éventuellement à trouver la politique optimale pour un environnement donné. Il existe un théorème qui arrange bien les choses car il relie les valeurs d'utilité locales $V^\pi(s)$ et les valeurs relatives de politiques entre elles. Il va donc être possible de s'appuyer sur les premières pour savoir dans quelle direction modifier les secondes afin de les améliorer. Nous ne saurions trop souligner l'importance de ce théorème, qui fonde la plupart des méthodes d'apprentissage par renforcement.

Théorème 16.1 (Relation d'ordre sur les politiques)

Soient π et π' deux politiques déterministes, telles que, pour tout état $s \in \mathcal{E}$:

$$Q^\pi(s, \pi'(s)) \geq V^\pi(s) \quad (16.12)$$

Alors la politique π' doit être au moins aussi bonne que la politique π , ce qui signifie que, pour tout état $s \in \mathcal{E}$:

$$V^{\pi'}(s) \geq V^\pi(s) \quad (16.13)$$

De plus, si l'inégalité large dans l'équation (16.12) est remplacée par un inégalité stricte, alors il en est de même dans l'équation (16.13).

Ce théorème indique donc que si l'on trouve une modification π' de la politique π qui vérifie l'équation (16.13), alors on obtient une meilleure politique. Concrètement, cela peut se traduire par une procédure d'amélioration de politique. Prenons en effet une politique déterministe π , et une autre politique déterministe π' identique à π , sauf pour un état s pour lequel: $\pi'(s) = a \neq \pi(s)$. Alors, pour tous les états autres que s , l'équation (16.12) est vérifiée. Si de plus nous avons $Q^\pi(s, \pi'(s)) \geq V^\pi(s)$, alors la nouvelle politique π' est meilleure que π .

Il est facile d'étendre cette procédure à une procédure qui améliore une politique π sur tous les états s . Pour cela, il suffit de prendre une politique π' qui pour chaque état s sélectionne

l'action qui semble la meilleure selon la fonction d'utilité $Q^\pi(s, a)$:

$$\begin{aligned}\pi'(s) &= \operatorname{ArgMax}_{a \in \mathcal{Z}} Q^\pi(s, a) \\ &= \operatorname{ArgMax}_{a \in \mathcal{Z}} E\{r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s, a_t = a\} \\ &= \operatorname{ArgMax}_{a \in \mathcal{Z}} \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V^\pi(s')]\end{aligned}\tag{16.14}$$

Cette procédure de type gradient⁶ choisit donc pour chaque état l'action qui semble la meilleure à un pas en avant, selon la fonction d'utilité V^π associée à la politique π . Il est facile de montrer que si la nouvelle politique π' choisie selon cette procédure n'est pas meilleure que la politique π , c'est que l'on a: $V^{\pi'} = V^\pi = V^*$, donc que l'on a atteint la politique optimale.

Tout ce qui précède a été obtenu pour le cas de politiques déterministes, mais peut être étendu sans problème au cas des politiques non déterministes. Nous reportons le lecteur intéressé à [SB98] par exemple.

16.3.3 Processus itératif d'amélioration de politique

La section précédente a montré comment passer d'une politique π décrite par sa fonction d'utilité V^π à une meilleure politique. Il est facile de voir comment on peut itérer ce processus. L'idée est d'alterner les phases d'évaluation de politique (section 16.3.1) avec les phases d'amélioration de politique (section 16.3.2). Nous pouvons alors obtenir une séquence de politiques en amélioration monotone:

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{A} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{A} \pi_2 \xrightarrow{E} \dots \xrightarrow{A} \pi^* \xrightarrow{E} V^*$$

où \xrightarrow{E} dénote une phase d'*évaluation* et \xrightarrow{A} une phase d'*amélioration*.

Cette procédure itérative converge en un nombre fini d'itérations vers la politique optimale si la politique est représentée par un processus markovien à nombre d'états fini⁷. Par ailleurs, la convergence observée empiriquement est généralement très rapide. Cependant, la phase d'évaluation de politique est très coûteuse puisqu'elle requiert de nombreux passages sur chaque état $s \in \mathcal{S}$ qui peuvent être très nombreux. Mais est-il possible de faire l'économie de la détermination précise des valeurs d'utilité relative à chaque politique π et π' que semble requérir théorème 16.1?

Il est heureusement envisageable de ne pas attendre la convergence de la phase d'*évaluation de politique* avant de lancer une nouvelle phase d'*amélioration de politique*. De fait, on dispose de théorèmes démontrant qu'il y aura convergence ultime sur la politique optimale même si l'alternance entre évaluation et amélioration se fait sur des granularités beaucoup plus fines que le processus alternatif décrit plus haut. Par exemple, on peut alterner une seule itération du processus d'évaluation de politique entre chaque phase d'amélioration, et dans ce cas, on obtient l'*algorithme d'itération de valeur*:

$$\begin{aligned}V_{k+1}(s) &= \max_{a \in \mathcal{Z}} E\{r_{t+1} + \gamma V_k(s_{t+1}) \mid s_t = s, a_t = a\} \\ &= \max_{a \in \mathcal{Z}} \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V_k(s')]\end{aligned}\tag{16.15}$$

6. Voir le chapitre 3.

7. Dans ce cas, en effet, le nombre de politiques est fini.

Cette équation de mise à jour à réaliser pour tout $s \in \mathcal{S}$, est en fait une version itérative de l'équation de Bellman (16.10) qui suppose que cette règle itérative converge vers un point fixe correspondant à la fonction d'utilité optimale $V^*(s)$.

L'action choisie à chaque instant est alors l'action maximisant le gain indiqué par la fonction d'utilité $V_k(s)$ courante :

$$\pi(s) = \operatorname{ArgMax}_{a \in \mathcal{Z}} [r' + \gamma V_k(s')]$$

On peut aussi utiliser l'équation 16.15 non pas en l'appliquant sur l'ensemble des états à chaque passe, mais état par état, de manière opportuniste, en fonction des situations rencontrées par l'agent. On montre qu'en général, si tous les états sont visités un nombre infini de fois, ce type de procédure asynchrone converge (en d'autres termes, cela signifie qu'il faut que quel que soit l'instant t considéré, chaque état soit visité après t). Il s'agit là d'une condition de convergence universelle pour les règles de mise à jour par propagation locale des informations. Cette remarque ouvre la possibilité d'apprendre en ligne en cours d'expérience, en mettant à jour les valeurs d'utilité des états rencontrés.

16.4 Si rien n'est connu : la méthode de Monte-Carlo

Les méthodes précédentes supposent que l'on connaisse l'environnement à travers des probabilités de transition d'états et de renforcement : $\mathcal{P}_{ss'}^a$ et $\mathcal{R}_{ss'}^a$. Si ce n'est pas le cas, le principe général est d'estimer ces valeurs par un échantillonnage obtenu en ligne, au cours des actions.

Les méthodes de Monte-Carlo supposent que l'on observe des *épisodes* complets⁸ et que l'on moyenne sur eux. Leur principe est d'estimer directement les valeurs d'utilité en calculant des moyennes de gain pour chaque état ou chaque paire (*état, action*) en fonction des expériences de l'agent.

Supposons que l'on cherche la valeur $V^\pi(s)$ pour l'état s et la politique π . Il suffit de considérer toutes les séquences d'états qui ont suivi l'état s et de calculer la moyenne des gains observés. On réalise alors empiriquement l'approximation de l'espérance : $V^\pi(s) = E_\pi\{R_t | s_t = s\}$. Sans entrer dans les détails, notons que cette méthode ne s'appuie plus sur la corrélation entre les états mise en évidence par l'équation de Bellman. Cette perte d'information nuit à la convergence du processus, mais permet en revanche de ne pas avoir à échantillonner uniformément tous les états et de s'apresantir sur les états les plus importants en ignorant les états inintéressants.

Afin d'améliorer la politique suivie, il est possible d'utiliser le même principe d'alternance de phase d'évaluation (selon la méthode ci-dessus) et de phase d'amélioration. Cependant, en raison de l'ignorance de l'environnement, il n'est plus possible de travailler avec l'équation (16.15). Il faut avoir recours à une procédure itérative portant sur les fonctions d'utilité $Q^\pi(s, a)$. On aura donc :

$$\pi(s) = \operatorname{ArgMax}_{a \in \mathcal{Z}} Q(s, a) \tag{16.16}$$

et une procédure d'amélioration itérative de politique :

$$\begin{aligned} Q^{\pi_k}(s, \pi_{k+1}(s)) &= Q^{\pi_k}(s, \operatorname{ArgMax}_{a \in \mathcal{Z}} Q^{\pi_k}(s, a)) \\ &= \max_{a \in \mathcal{Z}} Q^{\pi_k}(s, a) \\ &\geq Q^{\pi_k}(s, \pi_k(s)) = V^{\pi_k}(s) \end{aligned} \tag{16.17}$$

8. Des séquences d'états et d'actions s'arrêtant dans des états terminaux.

Pour plus de détails sur les conditions d'application de cette procédure, nous reportons le lecteur à [SB98].

16.5 Le meilleur des deux mondes : la méthode des différences temporelles

La méthode des différences temporelles (*temporal-difference learning*) combine des idées des méthodes issues de la programmation dynamique et des méthodes de Monte-Carlo. Comme les premières, elles prennent en compte les corrélations entre les états pour mettre à jour leur évaluation. Comme les secondes, elles n'ont pas besoin d'une connaissance *a priori* sur l'environnement. La méthode des différences temporelles s'appuie également sur une alternance de *phases d'évaluation* et de *phases d'amélioration*. Décrivons tour à tour les deux phases.

16.5.1 L'évaluation suivant la méthode des différences temporelles

Nous avons vu que l'estimation de la valeur d'utilité selon la méthode de Monte-Carlo repose sur une approximation de la formule :

$$V^\pi(s) = E_\pi\{R_t \mid s_t = s\}$$

qui se traduit par une opération itérative de mise à jour :

$$V(s_t) \leftarrow V(s_t) + \alpha [R_t - V(s_t)]$$

où R_t est le gain mesuré après l'instant t , et α est un paramètre constant. Cela peut se démontrer par la série d'égalités suivante. On suppose que l'espérance définissant la valeur de $V^\pi(s)$ est estimée par moyennage sur $m+1$ épisodes observés après passage par l'état s :

$$\begin{aligned} V_{m+1}^\pi(s) &= \frac{1}{m+1} \sum_{i=1}^{m+1} R_i \\ &= \frac{1}{m+1} \left(R_{m+1} + \sum_{i=1}^m R_i \right) \\ &= \frac{1}{m+1} (R_{m+1} + m V_m^\pi(s) + V_m^\pi(s) - V_m^\pi(s)) \\ &= \frac{1}{m+1} (R_{m+1} + (m+1) V_m^\pi(s) - V_m^\pi(s)) \\ &= V_m^\pi(s) + \frac{1}{m+1} [R_{m+1} - V_m^\pi(s)] \end{aligned}$$

L'intérêt de cette méthode de mise à jour incrémentale est qu'elle ne nécessite que la mémorisation de $V_m^\pi(s)$ et m et un calcul simple après chaque observation d'un épisode. On peut généraliser cette procédure à la forme générale suivante :

$$\text{NouvelleEstimation} \leftarrow \text{AncienneEstimation} + \alpha [\text{Cible} - \text{AncienneEstimation}] \quad (16.18)$$

dans laquelle $[\text{Cible} - \text{AncienneEstimation}]$ est une *erreur* sur l'estimation courante qui est réduite en allant d'un pas vers la *Cible*. Cette cible indique donc la direction dans laquelle aller. Elle peut être sujette à des variations aléatoires. Le pas α peut être une constante ou une variable décroissante lentement, ce qui est souvent utilisé pour stabiliser la valeur estimée.

La méthode de la programmation dynamique (voir l'équation (16.14)) utilise la formule :

$$V^\pi(s) = E_\pi \{ r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s \}$$

La méthode des différences temporelles est basée sur une mise à jour au coup par coup de cette estimation :

$$V_t^\pi(s) \leftarrow V_t^\pi(s) + \alpha [r_{t+1} + \gamma V_t^\pi(s_{t+1}) - V_t^\pi(s_t)] \quad (16.19)$$

On parle dans ce cas de *retour échantillonné* (*sample backup*) parce que la mise à jour s'effectue à partir d'observations individuelles obtenues durant l'action. Par contraste, les méthodes de *retour complet* (*full backup*) s'appuient sur une distribution complète des successeurs possibles. C'est le cas de l'équation d'estimation utilisée dans la programmation dynamique. La possibilité d'utiliser un retour échantillonné pour estimer incrémentalement les valeurs d'utilité est cruciale pour la faisabilité d'un apprentissage en ligne durant l'expérience de l'agent dans le monde. Cela conduit à l'algorithme 16.2.

Algorithme 16.2 Algorithme d'évaluation par la méthode des différences temporelles

Initialiser $V(s)$ arbitrairement, et π à la politique à évaluer.

faire

 pour chaque épisode

 Initialiser s

faire

 pour chaque étape de l'épisode

$a \leftarrow$ l'action donnée par π pour l'état s

 Exécuter l'action a ; recevoir le renforcement r ; et mesurer l'état suivant s'

$V^\pi(s) \leftarrow V(s) + \alpha [r + \gamma V^\pi(s') - V^\pi(s)]$

$s \leftarrow s'$

jusqu'à s est terminal

jusqu'à critère d'arrêt (convergence suffisante)

Cet algorithme offre la possibilité de mettre à jour les valeurs d'utilité tout en agissant dans le monde et en tirant parti des observations ainsi réalisées. Il faut cependant pour en assurer la convergence théorique que tous les états soient visités infiniment souvent sur un temps infini. Les propriétés et conditions de cette convergence sont encore du domaine de la recherche. (Pour les détails, on peut se reporter à [SB98]).

16.5.2 L'amélioration de politique avec les différences temporelles

Comme pour les méthodes de programmation dynamique et de Monte-Carlo, il faut préciser comment passer de l'évaluation de la politique à l'amélioration de la politique. Nous allons considérer deux approches typiques dont les principes sont de portée générale. La première est fondée sur une approche similaire à celles évoquées plus haut reposant sur une alternance de phases d'évaluation de politique et de phases d'amélioration. La seconde court-circuite d'une certaine manière l'idée même de politique.

16.5.3 SARSA : Une méthode d'amélioration « sur politique »

L'idée de base de l'algorithme SARSA ([SB98] section 6.4) est la suivante: à chaque choix d'action dans l'état courant s_t , l'agent suit approximativement (nous verrons pourquoi et comment) la politique courante π . Après observation du nouvel état courant s_{t+1} et du renforcement reçu r_{t+1} , il met à jour la valeur d'utilité de la situation rencontrée et est alors prêt à choisir l'action suivante. Il s'agit donc d'une méthode itérative alternant évaluation et amélioration. On qualifie ce genre d'approche de méthode *sur politique* (*on-policy*) car elle suppose que l'agent s'inspire à chaque instant de la politique courante π pour le choix de ses actions.

Afin de s'affranchir de la nécessité de connaître un modèle de l'environnement, l'approche SARSA estime la fonction d'utilité $Q^\pi(s, a)$ pour la politique courante et pour toutes les paires (*état, action*). La méthode des différences temporelles conduit alors à la mise à jour suivante:

$$Q^\pi(s_t, a_t) \leftarrow Q^\pi(s_t, a_t) + \alpha [r_{t+1} + \gamma Q^\pi(s_{t+1}, a_{t+1}) - Q^\pi(s_t, a_t)] \quad (16.20)$$

Cette mise à jour est effectuée après chaque transition partant d'un état s_t non terminal. Si s_{t+1} est un état terminal, alors $Q(s_{t+1}, a_{t+1})$ est défini comme égal à zéro. Comme cette règle de mise à jour utilise les valeurs de $s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}$, elle a été baptisée algorithme SARSA.

L'amélioration de politique se fait alors grâce à la procédure suivante. À chaque choix d'action, l'agent utilise les valeurs estimées $Q^\pi(s, a)$ pour sélectionner l'action à exécuter. Pour négocier le compromis exploitation contre exploration, l'agent utilise une procédure dite *ε -gloutonne* c'est-à-dire une méthode de gradient bruité: l'agent sélectionne en général l'action a_t associée à la meilleure valeur d'utilité $Q^\pi(s_t, a_t)$. De temps en temps cependant, avec une probabilité ε , il sélectionne aléatoirement une autre action, ce qui permet ainsi d'explorer les politiques possibles.

Algorithme 16.3 Algorithme d'évaluation itérative d'une politique

Donnée: la politique π à évaluer

Initialisation: $V(s) = 0$, pour tous $s \in \mathcal{E}^+$ {les états accessibles depuis s }

faire

$\Delta \leftarrow 0$

pour tout $s \in \mathcal{E}$ faire

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_a \pi(s, a) \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

fin pour

jusqu'à $\Delta < \theta$ {un petit nombre réel positif}

Sortie $V \approx V^\pi$

La procédure SARSA converge avec une probabilité de 1 vers une politique optimale si toutes les paires (*état, action*) sont visitées infiniment souvent sur une durée infinie et si le coefficient ε est bien réglé (par exemple, en posant $\varepsilon = \frac{1}{t}$).

16.5.4 Le *Q* – learning : Une méthode d'amélioration « hors politique »

L'algorithme du *Q-learning* ([Wat89]) utilise la dépendance explicite de la fonction d'utilité $Q(s, a)$ sur les actions, pour à la fois mettre à jour ces valeurs, donc converger vers la fonction

Algorithme 16.4 Algorithme SARSA d'amélioration de politique

```

Initialiser  $Q(s, a)$  arbitrairement.

faire
    pour chaque épisode
        Initialiser  $s$ 
        Choisir l'action  $a$  en utilisant une procédure  $\varepsilon$ -gloutonne dérivée des valeurs de  $Q^\pi(s, a)$ 
    faire
        pour chaque étape de l'épisode
            Exécuter l'action  $a$ ; recevoir le renforcement  $r$ ; et mesurer l'état suivant  $s'$ 
            Choisir l'action  $a'$  à partir de  $s'$  en utilisant une procédure  $\varepsilon$ -gloutonne dérivée des valeurs de  $Q^\pi(s, a)$ 
             $Q^\pi(s, a) \leftarrow Q^\pi(s, a) + \alpha [r + \gamma Q^\pi(s', a') - Q^\pi(s, a)]$ 
             $s \leftarrow s'; a \leftarrow a'$ 
        jusqu'à  $s$  est terminal
    jusqu'à critère d'arrêt (convergence suffisante)

```

optimale $Q^*(s, a)$, et aussi pour déterminer l'action à prendre dans la situation courante. Dans cette technique, la mise à jour des valeurs d'utilité se fait selon l'équation suivante:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma \max_{a \in \mathcal{Z}} Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (16.21)$$

tandis que l'action a sélectionnée dans l'état courant s est déterminée par une politique du genre ε -gloutonne assurant le compromis exploitation contre exploration. Il faut noter qu'il n'existe pas pour le moment de méthode générique pour résoudre ce compromis dans le cas du *Q-learning* et que la pratique est d'utiliser des règles *ad hoc* déterminées empiriquement pour chaque situation.

Comme pour toutes les procédures de mise à jour stochastiques dans des processus markoviens, la convergence vers la valeur optimale $Q^*(s, a)$ nécessite que chaque état soit visité infiniment souvent et que le paramètre α décroisse de manière appropriée.

Il est intéressant de noter que la convergence de la méthode est assurée quelle que soit la manière dont les actions sont sélectionnées à chaque instant, pourvu que tous les états soient visités infiniment souvent. C'est pourquoi on parle de méthode *hors politique (off-policy)*. En revanche, les vitesses de convergence observées sont souvent plus lentes que celles d'autres méthodes. Cela semble le prix à payer pour l'utilisation d'une méthode qui a révolutionné l'apprentissage par renforcement grâce à sa très grande aisance d'emploi et à la facilité d'établir des preuves à son propos.

16.5.5 TD(λ) : les méthodes de différences temporelles à plusieurs pas

Pour présenter les méthodes TD(λ), le mieux est de présenter d'abord le cas particulier TD(0). Cette méthode effectue ses mises à jour des valeurs d'utilité en ne regardant qu'un seul pas en avant. Même si cela suffit à garantir la convergence selon les conditions déjà soulignées, celle-ci peut être lente. Les méthodes TD(λ) la généralisent en effectuant des mises à jour selon un horizon plus lointain.

L'idée est la suivante: lorsqu'une information a été obtenue sur le renforcement r_{t+1} entre l'état courant s_t et le suivant s_{t+1} atteint avec l'action a_t , on peut mettre à jour la valeur $V(s_t)$ mais aussi, par ricochet, les valeurs des états antérieurement visités $V(s_{t-i})$. En général, on

ne met pas à jour uniformément les valeurs des états visités dans le passé, mais on utilise une pondération diminuant l'effet de la mise à jour au fur et à mesure que l'on remonte dans la séquence des états.

La formule générale de mise à jour s'écrit :

$$V(u) \leftarrow V(u) + \alpha [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] e(u) \quad (16.22)$$

où u est n'importe quel état pour lequel $e(u) \neq 0$. La *trace d'éligibilité* $e(u)$ détermine ainsi les états sujets à mise à jour. Une expression habituelle de trace d'éligibilité est :

$$e(s) = \sum_{k=1}^t (\lambda\gamma)^{t-k} \delta_{s,s_k}, \text{ où } \delta_{s,s_k} = \begin{cases} 1 & \text{si } s = s_k \\ 0 & \text{sinon} \end{cases} \quad (16.23)$$

avec $0 \leq \lambda < 1$ pour assurer la convergence.

L'éligibilité d'un état s définit ainsi le degré auquel il doit être sensible à la mise à jour actuelle. Quand $\lambda = 0$, on retrouve la méthode TD(0). Quand $\lambda = 1$, on retrouve à peu près la méthode de Monte-Carlo, c'est-à-dire que l'on met à jour chaque état à raison du nombre de fois où il a été visité durant l'épisode servant de base à la mise à jour.

Il est aussi possible de modifier en ligne la trace d'éligibilité :

$$e(s) \leftarrow \begin{cases} \gamma\lambda e(s) + 1 & \text{si } s = \text{l'état courant} \\ \gamma\lambda e(s) & \text{sinon} \end{cases}$$

L'algorithme TD(λ) est évidemment nettement plus coûteux à mettre en œuvre que TD(0), mais il converge en général beaucoup plus rapidement pour des valeurs de λ assez grandes (mais toujours < 1). Plusieurs travaux ont été consacrés à l'analyse des traces d'éligibilité. Elles peuvent être appliquées aux algorithmes de Q-learning, conduisant aux méthodes appelées Q(λ)-learning.

16.6 La généralisation dans l'apprentissage par renforcement

16.6.1 Le problème

Jusqu'à présent nous avons implicitement supposé que les états et les actions étaient énumérables et qu'il était possible de représenter les différentes fonctions (par exemple la fonction d'utilité V) par des tables de valeurs. Or, excepté pour des environnements très simples, cela implique des tailles de mémoire irréalistes. De plus, dans de nombreuses applications, l'espace des états, et parfois aussi celui des actions, est continu, rendant impossible l'usage direct de tables. Finalement, les algorithmes de mise à jour de tables font une utilisation assez peu efficace de l'information glanée en cours d'expérience dans la mesure où, alors qu'il est fréquent que des états similaires aient des valeurs d'utilité et des actions optimales attachées similaires, la modification d'une valeur dans la table n'entraîne pas celle des autres. Dans ces conditions, l'utilisation de tables semble peu opportun et il faut chercher une représentation des politiques à la fois plus compacte et permettant une utilisation plus efficace de l'information.

Une idée assez immédiate est d'avoir recours à des techniques de généralisation dans l'espace \mathcal{E} des états, et éventuellement dans l'espace \mathcal{Z} des actions. De cette manière, lorsqu'une situation impliquant un état et une action donnés est observée, l'information obtenue à cette occasion peut être transférée par généralisation aux états et actions similaires. Il devient alors envisageable d'appliquer l'ensemble des techniques d'induction supervisée à ce problème.

Les fonctions que l'on peut vouloir apprendre incluent :

- l'apprentissage direct d'une politique $\pi: \mathcal{E} \rightarrow \mathcal{Z}$;
- l'apprentissage de la fonction d'utilité $V: \mathcal{E} \rightarrow \mathbb{R}$;
- l'apprentissage de la fonction d'utilité $Q: \mathcal{E} \times \mathcal{Z} \rightarrow \mathbb{R}$;
- l'apprentissage de l'environnement :
 - la fonction de transition :
 - déterministe: $\mathcal{E} \times \mathcal{Z} \rightarrow \mathcal{E}$,
 - ou non déterministe: $\mathcal{E} \times \mathcal{Z} \times \mathcal{E} \rightarrow [0, 1]$,
 - la fonction de renforcement: $\mathcal{E} \times \mathcal{Z} \rightarrow \mathbb{R}$.

Certaines de ces fonctions, à savoir la fonction de transition et la fonction de récompense, sont du ressort direct de nombreuses méthodes classiques d'apprentissage supervisé, telles des méthodes connexionnistes (chapitre 10), l'induction d'arbres de décision (chapitre 11) ou des méthodes par k -plus proches voisins (chapitre 14). Il est en effet facile d'obtenir de nombreux exemples, et la fonction cible est généralement fixe ou peu changeante (sauf si l'environnement est gravement perturbé). D'autres posent plus de problèmes, comme la fonction de politique car il est difficile d'obtenir des exemples à partir desquels apprendre. Dans tous les cas, il est souhaitable de disposer de méthodes d'apprentissage qui permettent la prise en compte incrémentale des exemples au fur et à mesure que l'exploration du monde les rend disponibles et qui ont la capacité de suivre une fonction cible changeante.

16.6.2 Généralisation par approximation de la fonction d'utilité

Nous supposons dans un premier temps que le problème de prédiction porte sur la fonction d'utilité sur les états : V . L'idée est d'affiner une estimation de la fonction optimale V^* par des itérations successives d'une estimation V_t au temps t en fonction de l'expérience courante de l'agent dans le monde. On suppose ici que les fonctions estimées V_t ne sont plus des tables de valeurs mais font partie d'une classe de fonctions que l'on supposera paramétrée par un vecteur θ . Cela signifie que la fonction estimée V_t dépend entièrement de θ_t variant d'un instant t au suivant. L'une des nouveautés par rapport à un problème de régression classique est que l'apprentissage s'opère ici *en ligne* avec des exemples issus d'une distribution dépendant de l'action de l'agent et non d'une distribution d'exemples indépendamment et identiquement distribués (i.i.d.) comme c'est le cas général en induction.

Typiquement, l'estimation V_t peut être réalisée par un réseau connexionniste dont les poids sont réglés en cours d'apprentissage, ou par un arbre de décision. Dans ce dernier cas, le vecteur θ_t décrit les branchements de l'arbre appris.

Les exemples servant à l'apprentissage dépendent des méthodes de prédiction de gain utilisées. Par exemple, il est courant de prendre l'estimation de gain calculée selon la méthode des différences temporelles : $s \mapsto r_{t+1} + \gamma V_t(s_{t+1})$.

L'utilisation de méthodes de généralisation pose plusieurs questions :

1. L'application d'une méthode de généralisation dans l'espace des états signifie que l'observation d'une situation ou d'une séquence de situations particulière entraîne la modification de l'estimation de l'utilité non seulement pour la situation concernée mais aussi pour d'autres situations. Existe-t-il des garanties que cette méthode converge ? Et si oui, converge-t-elle vers la fonction d'utilité optimale, V^* dans le cas de la fonction d'utilité définie sur les états ?
2. Est-ce que les méthodes itératives d'amélioration de politique entremêlant phases d'évaluation et phases d'amélioration peuvent encore s'appliquer avec des garanties de convergence vers la politique optimale ?

3. Est-ce que les techniques d'exploration ε -gloutonnes continuent à être efficaces pour, d'une certaine manière, échantillonner l'espace des exemples ?
4. Quelle mesure d'erreur utiliser pour évaluer la performance de l'approximation de la fonction d'utilité ? Est-il encore approprié d'utiliser la mesure d'erreur quadratique qui est employée pour la régression ?

Au moment de la rédaction de cet ouvrage, ces questions font l'objet de recherches et n'ont pas encore de réponses définitives. Répondre aux deux premières questions est d'autant moins facile que l'apprentissage par renforcement implique souvent à la fois un apprentissage de type incrémental capable de prendre en compte les exemples au fur et à mesure de leur arrivée, mais aussi un environnement qui peut évoluer au cours du temps. De plus, sans même avoir affaire à un environnement changeant, les exemples eux-mêmes évoluent du fait qu'ils correspondent souvent à des évaluations de gain qui sont adaptatives, comme c'est le cas par exemple pour la méthode des différences temporelles. Les problèmes de convergence et de vitesse de convergence sont donc encore plus aigus que pour l'induction classique.

De nombreux travaux ont été publiés concernant des expériences d'apprentissage par renforcement avec généralisation : Boyan et Moore [BM95a] ont utilisé des méthodes de plus proches voisins dans une approche d'itération de valeur ; Lin [Lin91] a mis en œuvre un réseau connexionniste avec apprentissage par rétropropagation de gradient pour apprendre la fonction d'utilité $Q(s, a)$ dans le Q-learning ; Watkins [Wat89], toujours dans le cadre du Q-learning, a utilisé la technique de CMAC (*Cerebellar Model Articulatory Controller*) due à Albus [Alb75, Alb81]) (voir plus loin la section 16.6.3) et a été suivi par de nombreux autres chercheurs ; Tesauro [Tes95a] a utilisé un réseau connexionniste pour apprendre la fonction $V(s)$ dans le cas du jeu de Backgammon et Zhang et Dietterich [ZD95] ont utilisé un réseau connexionniste dans une technique de différence temporelle $TD(\lambda)$ pour apprendre des stratégies d'ordonnancement de tâches pour des ateliers.

Dans l'ensemble, même si des résultats spectaculaires ont été rapportés, des interférences pernicieuses sont observées entre la mise à jour adaptative des valeurs d'utilité et l'apprentissage par généralisation. En effet, alors que dans les environnements discrets avec maintien de tables de valeurs, il existe des garanties que toute opération qui met à jour la valeur d'utilité (selon les équations de Bellman) ne peut que réduire l'erreur entre la valeur courante estimée et la valeur optimale, de telles garanties n'existent plus dans le cas continu avec des techniques de généralisation. Il semble en particulier que les méthodes de différences temporelles à plusieurs pas ne soient pas appropriées dans les méthodes de généralisation, et qu'il faille réexaminer les méthodes d'évaluation locales de valeur d'utilité dérivées des équations de Bellman. Boyan et Moore, par exemple, ont les premiers attiré l'attention sur ce problème en 1995 [BM95a] en donnant des exemples de fonctions d'utilité dont l'erreur croissait de manière arbitrairement grande par l'utilisation de techniques de généralisation. Certaines solutions *ad hoc* pour des classes de problèmes particulières ont été proposées, mais elles ne conduisent généralement qu'à des optima locaux. La question des conditions nécessaires et suffisantes pour la convergence dans le cas de l'utilisation de méthodes de généralisation de fonctions d'utilité reste donc ouverte.

Parmi les causes potentielles de problèmes figure le fait que lors de l'apprentissage par renforcement, les exemples utilisés pour l'apprentissage résultent d'un échantillonnage qui reflète la politique courante, laquelle n'est pas la politique optimale cible. Il semble que les méthodes d'exploration classiques, telles que la méthode ε -gloutonne, ne soient pas adaptées à cette situation. Là encore, des recherches sont nécessaires pour mieux comprendre les interactions entre échantillonnage, apprentissage et amélioration de politique.

Finalement, la question de la mesure d'erreur à utiliser pour la généralisation a été soulevée. En effet, contrairement à la tâche de régression pour laquelle l'écart quadratique est approprié,

l'apprentissage par renforcement vise moins à approximer correctement les fonctions d'utilité qu'à fournir la meilleure politique. Ce qui compte n'est pas l'écart à la fonction optimale d'utilité, mais le fait que l'estimation d'utilité conduise bien à ce que la meilleure politique soit choisie. Il s'agit donc de respecter l'ordre relatif des politiques afin que la meilleure, π^* , soit en tête. Nous verrons dans la section 16.6.4 que cela a conduit récemment à réexaminer l'ensemble du problème.

Étant données toutes ces interrogations, les deux approches par estimation de fonction les plus employées actuellement reposent toutes les deux sur une technique de descente de gradient pour réduire l'écart quadratique entre les valeurs estimées à l'instant t et la cible courante, par exemple celle rapportée par la méthode de différences temporelles.

- La première approche consiste à utiliser un *réseau connexionniste* (généralement un perceptron multicouche) comme réalisation d'un vecteur de paramètre $\boldsymbol{\theta}_t$ en le modifiant par une technique de rétropropagation de gradient selon une formule telle que la suivante (ici pour la fonction d'utilité V) :

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha [v_t - V_t(s_t)] \nabla_{\boldsymbol{\theta}_t} V_t(s_t) \quad (16.24)$$

où v_t représente la cible, c'est-à-dire l'estimation courante de $V^\pi(s_t)$, et α est un pas d'apprentissage décroissant.

- La seconde approche consiste à utiliser une *combinaison linéaire de n fonctions de base* pour approximer la fonction cible. L'ensemble des fonctions de base $\Phi(s) = \{\phi_1(s), \dots, \phi_n(s)\}$ est défini sur l'espace \mathcal{E} des états. L'approximation de la fonction d'utilité correspond à la formule :

$$V_t(s) = \boldsymbol{\theta}_t^\top \Phi(s) = \sum_{i=1}^n \theta_t(i) \phi_i(s) \quad (16.25)$$

Dans ce cas, le gradient de la fonction V_t par rapport à $\boldsymbol{\theta}_t$ devient : $\nabla_{\boldsymbol{\theta}_t} V_t(s) = \Phi(s)$. Cette règle a l'avantage d'être simple et de conduire à un seul optimum. Comme de plus elle est assez efficace en termes de données et de calculs, elle a la faveur de nombreux chercheurs et praticiens.

Une approche duale consiste à apprendre non pas la combinaison de fonctions de base, mais à apprendre ces fonctions de base elles-mêmes. Souvent cela correspond à apprendre des voisinages dans l'espace. Ces méthodes font l'objet de la section suivante.

16.6.3 Méthodes de généralisation par partition de l'espace

L'idée de ces méthodes est de décider de la réponse en un point (par exemple l'utilité associée à un état, ou à un couple (état, action)) en fonction des valeurs connues dans son voisinage. L'apprentissage consiste dans ce cas à définir le voisinage en chaque point. Le chapitre 14 traite ce problème en général, nous indiquons donc ici seulement les grandes approches testées en apprentissage par renforcement.

- *Techniques par couverture de voisinages uniformes.* L'idée est d'associer à chaque point dont la valeur est connue un voisinage, par exemple une boule (voir la figure 16.5). La valeur d'un point inconnu est décidée en fonction des voisinages, par exemple des boules, dont elle fait partie. On utilise souvent une fonction linéaire des valeurs des boules dont le point fait partie. On peut modifier la densité des boules, ou bien leur forme, ce qui revient alors à rendre non linéaire l'approximation.

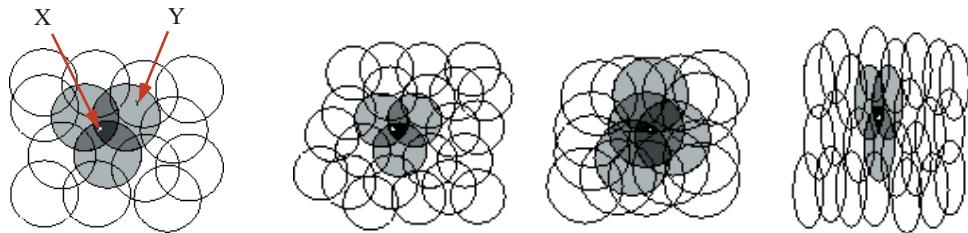


FIG. 16.5 – La valeur cherchée au point X (par exemple $V(X)$) dépend des valeurs des points Y dont les voisinages incluent X (d'après ([SB98], p.203)).

- *Techniques par partition adaptative de l'espace.* Une idée qui a fait l'objet de travaux récents consiste à définir adaptativement une partition de l'espace. Dans ce cas, chaque partition est exclusive des autres, et la valeur d'un point est celle associée à la partition à laquelle il appartient (voir figure 16.6). La granularité de la partition est d'autant plus fine que la valeur estimée connaît des variations importantes, dans un esprit similaire à une décomposition en éléments finis. Le lecteur est invité à se reporter par exemple à [MA95, MM99, Rey00].

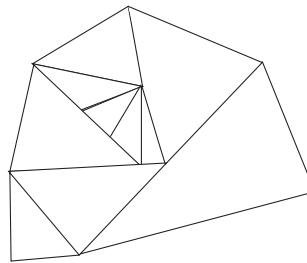


FIG. 16.6 – Sur cet exemple où l'on suppose que l'espace des états est bidimensionnel, la partition de l'espace est à granularité variable, d'autant plus fine que les variations de la fonction estimées sont importantes.

- *Techniques par utilisation de fonctions noyau.* Les fonctions noyau permettent de définir un voisinage variable autour des points. Une fonction radiale typique est la fonction gaussienne dont la variable est la distance entre le centre c et le point s :

$$\phi_i(s) = \exp\left(-\frac{\|s - c_i\|^2}{2\sigma_i^2}\right) \quad (16.26)$$

Ces fonctions de voisinage présentent l'avantage d'être adaptatives et différentiables, d'où leur emploi fréquent.

- *Approches hiérarchiques.* Une approche pour essayer de faire face à la malédiction de la dimensionalité est de les traiter comme des hiérarchies de problèmes d'apprentissage avec des granularités croissantes. Par exemple, l'une des méthodes employées consiste à utiliser une hiérarchie sous forme de porte logique (voir figure 16.7). Chaque boîte reçoit une description de l'environnement et prescrit un comportement, la porte choisit lequel est effectivement appliqué. L'apprentissage peut alors consister à apprendre les boîtes ou bien le critère de sélection de la porte.

En allant plus loin, il devient intéressant de voir s'il est possible de décomposer un comportement en tâches indépendantes, ou quasi indépendantes, et de structurer un problème

en fonction des interrelations entre sous-problèmes. Cela ouvre la voie à l'intégration de plusieurs comportements, qu'ils soient issus de plusieurs types ou niveaux de descriptions, ou qu'ils résultent d'univers multiagent. L'article [Die00] constitue un bon point d'entrée pour la littérature consacrée à cette approche.

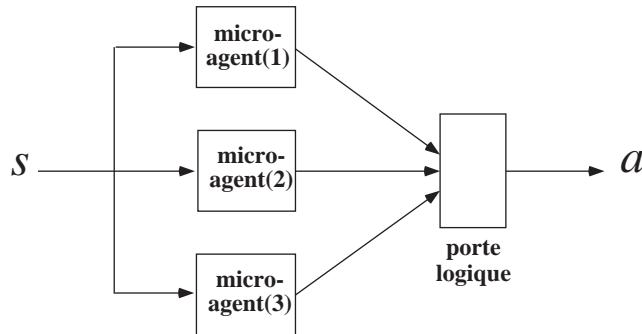


FIG. 16.7 – Une hiérarchie organisée sous forme de porte logique.

16.6.4 Méthodes directes d'apprentissage de politique

Les difficultés liées aux méthodes de généralisation pour approximer les fonctions d'utilité, concernant en particulier la convergence, non garantie, et la qualité de l'approximation, parfois très mauvaise, ont relancé l'idée d'opérer une recherche directe dans l'espace des politiques plutôt que de passer par l'apprentissage de fonctions d'utilité. Au lieu de chercher à minimiser un écart quadratique par rapport à la fonction d'utilité vraie, le principe consiste à exprimer les politiques comme des fonctions paramétrées, par un vecteur θ , et à chercher une valeur de θ correspondant à une politique optimale, c'est-à-dire maximisant l'espérance de gain $E_\pi(R_t)$.

Pour ce faire, on passe généralement par une expression paramétrée par θ de l'estimation de gain, noté $\eta(\theta)$, et on opère ensuite une descente de gradient pour trouver la valeur de θ maximisant cette expression. Les difficultés de cette approche sont nombreuses. La première est d'évaluer précisément l'espérance de gain $\eta(\theta)$. La seconde consiste à trouver une paramétrisation pratique de la politique, et donc du gain. Un réseau connexionniste multicouche est souvent utilisé à cet effet. La troisième concerne la possibilité d'utiliser une technique de descente de gradient sur $\eta(\theta)$. Cette fonction est rarement différentiable, et il faut donc avoir recours à des astuces permettant de reformuler le problème. Finalement se pose l'éternel problème de la combinaison entre évaluation de l'espérance de gain et amélioration de la politique. L'approche par recherche directe de politique n'est donc pas évidente, mais elle connaît actuellement un vif intérêt (voir par exemple [BM99, Bax00, GU00, Wil92b]).

16.7 Le cas des environnements partiellement observables

Dans de nombreuses situations issues du monde réel, l'agent ne peut pas avoir une perception parfaite et complète de l'état du monde, mais doit faire face à toute une variété d'incertitudes. Le modèle des processus markoviens est alors inapplicable tel quel. Une idée est d'essayer de remédier au manque instantané d'information par la prise en compte d'une mémoire des événements passés. Des modèles à base de réseaux connexionnistes récurrents ont été expérimentés, mais sans qu'ils soient concluants au-delà de problèmes simples. Une autre approche consiste à prendre explicitement en compte l'incertitude sur les états. C'est ce que permettent les modèles

de Markov cachés. Les techniques développées dans le chapitre 13 sont des candidates naturelles pour réaliser un tel apprentissage.

16.8 Exemples d'application

Il n'est pas question de fournir un panorama complet des domaines d'application des méthodes d'apprentissage par renforcement. Nous nous contenterons donc de présenter rapidement quelques exemples pour lesquels il est facile de trouver davantage de détails dans la littérature scientifique.

16.8.1 Le TD-Gammon

Bien que l'apprentissage par renforcement ne se prête pas naturellement au domaine du jeu car l'environnement, commandé par l'adversaire, n'est pas stationnaire, il y a eu de nombreuses tentatives de systèmes d'apprentissage pour des jeux divers. La plus ancienne application réussie est celle du jeu de dames américain⁹ due à Samuel en 1959 [Sam59]. Le programme apprenait une fonction d'évaluation $V(s)$ représentée par une fonction linéaire d'un certain nombre de facteurs déterminés par Samuel. Il employait un mécanisme d'apprentissage similaire à celui de l'algorithme d'itération de valeur, des différences temporelles et du Q-learning.

Une autre succès plus récent est celui de Tesauro dans le domaine du backgammon [Tes92, Tes94a, Tes95a]. Ce jeu comporte environ 10^{20} états, ce qui rend impossible une méthode basée sur une table d'états. Il faut donc utiliser une méthode de généralisation dans l'espace des états. Tesauro a employé une perceptron multicouche (voir chapitre 10) à une couche cachée avec apprentissage par rétropropagation de gradient, pour réaliser un système d'estimation de la fonction de valeur :

$$\text{Position sur le jeu} \longrightarrow \text{Probabilité de victoire pour le joueur courant}$$

Une première version de base de l'algorithme appelé TD-GAMMON ne comportait aucune connaissance spécifique du domaine, tandis que les versions ultérieures (TD 1.0, TD 2.0, TD 2.1) utilisaient des connaissances propres à certaines positions de jeu. Pour toutes ces versions, l'apprentissage fut réalisé par simulation de jeu de l'ordinateur contre lui-même. Remarquablement, aucune stratégie d'exploration n'était utilisée et l'algorithme choisissait toujours le coup apparemment le meilleur. Cela ne pose pas de problème au backgammon car les situations de jeu obtenues dépendent en partie d'un tirage aux dés ce qui suffit à garantir que tous les états seront visités. De plus, il s'agit d'un jeu dans lequel les parties terminent en un temps fini, ce qui assure que des renforcements sont reçus assez fréquemment. Les résultats obtenus par les différentes versions de TD-GAMMON sont résumés dans le tableau suivant :

	Parties jouées en apprentissage	Cellules sur la couche cachée	Résultats
Version de base			Médiocre
TD 1.0	300 000	80	Battu de 13 points en 51 matches
TD 2.0	800 000	40	Battu de 7 points en 38 matches
TD 2.1	1 500 000	80	Battu d'un point en 40 matches

9. Contrairement au jeu de dames joué en Europe continentale, le jeu de dames nord-américain (*checkers*) se joue sur un damier de 8x8 cases. De plus certaines règles de prise sont différentes.

TD-GAMMON se place parmi les meilleurs joueurs mondiaux.

16.8.2 Applications au contrôle et à la robotique

Si les applications de l'apprentissage par renforcement sont de plus en plus nombreuses, elles restent encore du domaine de l'art autant que de la science. C'est pourquoi il est intéressant avant de se lancer dans un problème d'étudier ce qui a été fait pour des problèmes connexes. Sans chercher, ni pouvoir, être exhaustifs, nous citons ici des travaux remarquables dans le domaine du contrôle et de la robotique. Les références associées fournissent un point de départ pour explorer la littérature sur ce thème.

Le contrôle et la robotique se prêtent bien à l'apprentissage par renforcement car il s'agit d'applications souvent difficiles à programmer complètement, dans lesquelles l'information n'est disponible que lors du fonctionnement de manière incrémentale et dans un environnement parfois changeant. Parmi les applications les plus spectaculaires, on compte le robot jongleur de Schaal et Atkeson [Sch94]. Ce robot comporte deux bras commandés par un système à trois degrés de liberté. Il prend une décision d'action toutes les 200 ms et doit tenter de maintenir en l'air une sorte de pendule inversé.

Une autre application concerne des robots mobiles devant pousser des boîtes d'un endroit à un autre dans des pièces ([MC91]). Ce problème se caractérise par de grandes incertitudes sur l'effet des actions. L'approche utilisée employait une décomposition hiérarchique des tâches en sous-tâches. Une application similaire ([Mat94]) impliquait quatre robots mobiles devant rassembler des disques. Outre l'immense espace d'états impliqués, la tâche se caractérise par des problèmes de contrôle distribué, de communication et, éventuellement, par le partage des connaissances apprises.

L'apprentissage par renforcement a été également employé avec succès dans une tâche de contrôle de plusieurs ascenseurs dans un immeuble de dix étages. L'objectif est de minimiser la moyenne du temps d'attente des utilisateurs. L'approche utilisée avec du Q-learning et un estimateur dans l'espace des états à l'aide d'un réseau connexionniste a donné d'excellents résultats comparés à l'état de l'art.

D'autres applications incluent l'optimisation du remplissage de containers, l'allocation dynamique de canaux pour les téléphones cellulaires ([SB97]) et l'ordonnancement de tâches pour l'installation et le test de charges pour la navette spatiale ([ZD95, Zha96]).

16.9 Bilan et perspectives

L'apprentissage par renforcement s'intéresse au problème général se posant à un agent devant apprendre à choisir ses actions dans le but d'accroître son espérance de gain à long terme. La structure de son environnement étant généralement supposée inconnue, l'agent doit apprendre à partir de ses interactions avec le monde.

Dans les approches fondées sur les fonctions d'utilité, l'agent cherche à apprendre l'utilité de chaque *état* ou de chaque paire (*état, action*). Il sélectionne alors l'action associée à l'utilité maximale. Si la fonction d'utilité estimée est exacte, cette approche conduit à la politique optimale sous des conditions très générales ([SB98, BT96]). Cependant, pour la plupart des problèmes du monde réel, il est impossible de représenter les fonctions d'utilité exactement, en particulier avec des tables de valeurs. L'agent doit alors chercher une bonne approximation de la fonction d'utilité au sein d'une classe restreinte de fonctions (par exemple sous la forme d'un réseau connexionniste ou d'une classe de fonctions noyau). Cette approche a permis l'obtention de succès remarquables dans l'apprentissage de jeux (jeu de dames [Sam59], backgammon

[Tes92, Tes94a], jeu d'échecs [BTW00]), dans le domaine de l'ordonnancement de tâches [ZD95] et dans l'allocation dynamique de canaux de communication [SB97]).

La combinaison entre évaluation et apprentissage par généralisation pose de sérieux problèmes non encore résolus. C'est pourquoi la recherche directe dans l'espace des politiques est une option qui retient l'attention des chercheurs. Il faut signaler également les efforts visant à rendre moins empiriques les méthodes d'exploration utilisées pour échantillonner les situations (voir [Str00]).

Prolongeant l'apprentissage hors politique, certains travaux actuels se penchent sur l'apprentissage lorsque l'expérience porte sur un environnement qui diffère de l'environnement cible. C'est le cas par exemple de l'apprentissage de la bicyclette qui se fait avec une bicyclette à stabilisateurs pour lequel les contraintes sont différentes (voir [Ran00]). Ce sera le cas éventuellement de robots d'exploration planétaire.

Par ailleurs, une question essentielle porte sur l'intégration de l'apprentissage par renforcement, qui est de fait un apprentissage de réflexes, avec l'activité de planification qui implique un raisonnement de nature beaucoup plus stratégique.

Notes historiques et sources bibliographiques

L'apprentissage par renforcement a une longue histoire et plusieurs ascendances. L'une d'entre elles concerne les théories behavioristes de l'apprentissage par essais et erreurs, association et punitions-récompenses. Une autre réside dans les théories du contrôle optimal et des approches par programmation dynamique, en particulier dues à Bellman. Une autre est directement liée aux efforts en intelligence artificielle pour simuler des souris cybernétiques, apprendre à jouer au tic-tac-toe ou aux dames, ou encore modéliser certains apprentissages au niveau neuronal.

L'idée d'utiliser une fonction associant à chaque couple (état, action) une estimation de sa valeur remonte à Shannon [Sha50] en 1950 qui la proposa dans le cadre du jeu d'échec. L'un des premiers articles influents en intelligence artificielle est celui de Minsky en 1961 [Min61] dans lequel il introduit le problème du *credit assignment problem* central en apprentissage par renforcement. Donald Michie aussi, un disciple de Turing, explora plusieurs méthodes d'apprentissage par renforcement dont le système BOXES [MC68] qui, dans le cadre du pendule inversé, associe une action à chaque « boîte » dans l'espace des états. Nous citerons également Harry Klopf, qui dans les années 1970 fût l'un de ceux qui insistèrent sur la différence entre apprentissage supervisé et apprentissage par renforcement. Il introduisit les premiers éléments de l'idée d'apprentissage par différence temporelle, idée qui fût reprise et développée par Barto et Sutton dans les années 1980 et 1990. Watkins en 1989 [Wat89] réunit les approches de la théorie du contrôle et de l'apprentissage par différence temporelle dans le Q-learning qui eut un grand impact sur le domaine etaida à la propagation des idées d'apprentissage par renforcement dans des cercles plus larges d'utilisateurs et de théoriciens, aidé en cela par le succès de Tesauro sur le back-gammon. La dernière décennie a connu un développement exceptionnel de ce domaine de recherche qui est certainement appelé à connaître d'autres révolutions.

Le compromis exploitation contre exploration est connu depuis longtemps et a été modélisé par le scénario du bandit à deux bras par exemple par [Bel61] qui en a fait une analyse extensive dans le contexte de la théorie de la décision et du contrôle adaptatif. Holland ([Hol75]) l'a étudié également dans le cadre de l'analyse des algorithmes génétiques afin de montrer que ceux-ci réalisent spontanément une allocation optimale entre exploration et exploitation. On peut aussi se référer à [BF85].

Le livre de Barto et Sutton [SB98] est un ouvrage irremplaçable pour l'étude de l'apprentissage par renforcement et pour les sources historiques et bibliographiques le concernant.

Résumé

L'apprentissage par renforcement concerne l'apprentissage par un agent autonome d'une politique optimale, c'est-à-dire de l'action la mieux adaptée à chaque situation envisageable pour le système décisionnel considéré. La structure de son environnement étant généralement supposée inconnue, l'agent doit apprendre à partir de ses interactions avec le monde. En particulier, aucun professeur ne lui dit quelle action est la meilleure à prendre dans une situation donnée et seul un signal de renforcement assez pauvre (un scalaire) l'informe de temps en temps de sa performance liée à ses décisions passées.

Classiquement, l'apprentissage par renforcement est basé sur une fonction d'utilité. Divers algorithmes et structures de représentations de l'environnement ont été proposés pour apprendre cette fonction d'utilité dans le cadre formel des processus décisionnels markoviens (PDM).

Dans ces approches, l'agent cherche à apprendre l'utilité de chaque *état* ou de chaque paire (*état, action*). Il sélectionne alors l'action associée à l'utilité maximale. Si la fonction d'utilité estimée est exacte, cette approche conduit à la politique optimale sous des conditions très générales ([SB98, BT96]). Cependant, pour la plupart des problèmes du monde réel, il est impossible de représenter les fonctions d'utilité exactement, notamment avec des tables de valeurs. L'agent doit alors chercher une bonne approximation de la fonction d'utilité au sein d'une classe restreinte de fonctions (par exemple sous la forme d'un réseau connexionniste ou d'une classe de fonctions noyau). La difficulté est de trouver une représentation compacte assurant la convergence des méthodes d'apprentissage pour les PDM. C'est pourquoi de nouvelles approches d'apprentissage plus direct de la politique sont aussi explorées.

Cinquième partie

**Approfondissements et annexes
techniques**

Chapitre 17

Approfondissement sur l'analyse de l'induction

Le chapitre 2 a exposé les notions et les principes de base permettant d'aborder les méthodes développées en apprentissage artificiel et présentées dans cet ouvrage. Certaines d'entre elles cependant se réfèrent à une analyse théorique plus poussée, s'appuyant en particulier sur les travaux de Vapnik. Ceux-ci sont présentés ici ainsi que les approches relevant de principes de contrôle de l'espace d'hypothèses : principe de minimisation du risque structurel, théorie de l'estimation bayésienne, théorie de la régularisation, principe de compression de l'information. Le lecteur trouvera ainsi un complément d'information utile pour aller plus loin dans l'étude de ces méthodes.

Par ailleurs, il est essentiel que les praticiens de l'apprentissage connaissent l'un des théorèmes fondamentaux de l'induction : le no-free-lunch theorem qui délimite le pouvoir inductif de n'importe quelle méthode d'apprentissage en montrant qu'elles ne peuvent être universelles. Une confrontation avec l'analyse de Vapnik aide à comprendre la portée de cette analyse et ses conséquences pratiques pour l'apprentissage.

Ce chapitre se termine par un panorama non exhaustif mais évocateur de nouvelles directions de recherche visant à dépasser certaines des limitations du paradigme dominant en apprentissage artificiel.

17.1 L'analyse de l'induction de Vapnik

Le chapitre 2 a introduit les grands principes inductifs : principe de minimisation du risque empirique, principe de la décision bayésienne, principe de compression de l'information, et les approches dérivées de sélection de modèles qui prennent en compte l'espace d'hypothèses. Ces principes, intuitivement raisonnables, offrent-ils des garanties de performance ? Pour ce qui est du principe de minimisation du risque empirique (*Empirical Risk Minimization : ERM*), l'étude théorique rapportée dans le chapitre 2 avait montré l'intérêt d'une « analyse dans le pire cas », valable quelle que soit la distribution des exemples et pour n'importe quelle fonction cible. Cette analyse *PAC* (apprentissage Probablement Approximativement Correct), menée dans le cas d'espaces de fonctions indicatrices de cardinal fini et pour des fonctions de perte comptant le nombre d'erreurs de classification, a conduit à l'utilisation d'une convergence uniforme, s'appuyant sur la preuve que pour toutes les hypothèses de l'espace \mathcal{H} , il y a convergence du risque empirique mesuré sur l'échantillon d'apprentissage et du risque réel. Est-il possible de généraliser cette étude ? C'est ce que des travaux, dus en grande partie à Vladimir Vapnik, ont permis de réaliser.

Il est important de les connaître car ils ont débouché sur des concepts de pouvoir heuristique puissant (la dimension de Vapnik-Chervonenkis) et sur de nouveaux algorithmes généraux et performants (les séparateurs à vastes marges et leurs dérivés (SVM), voir le chapitre 9)

Nous allons aborder successivement le cas d'espaces d'hypothèses indicatrices de cardinal infini, puis celui d'espaces de fonctions quelconques avec des fonctions de perte également quelconques.

17.1.1 Cas où $|\mathcal{H}| = \infty$ et $\mathcal{F} \subseteq \mathcal{H}$

Dans le cas où le nombre d'hypothèses est fini, on comprend que l'on puisse borner la probabilité que l'on trouve parmi elles une hypothèse apparemment correcte sur l'échantillon d'apprentissage et pourtant médiocre en général. Lorsque la classe d'hypothèses contient un nombre infini d'éléments, il semble que l'on ne puisse pas dire grand chose. Pourtant, intuitivement, on sent qu'il doit pouvoir exister des classes d'hypothèses de cardinal infini mais d'expressivité limitée, pour lesquelles la performance sur l'échantillon d'apprentissage peut servir d'indicateur de la performance réelle. Nous allons voir dans cette section une caractérisation possible de cette diversité qui peut servir à établir le lien entre risque empirique et risque réel.

L'idée générale est la même que pour le cas où \mathcal{H} est de cardinalité finie (voir section 2.3 dans le chapitre 2). Il s'agit d'essayer de borner la probabilité d'avoir une hypothèse apparemment bonne, c'est-à-dire ici de risque empirique nul (hypothèse consistante) sur l'échantillon d'apprentissage, qui soit en fait de risque réel $> \varepsilon$:

$$P_{\mathcal{D}_{\mathcal{Z}}} \{ \mathcal{S} : \exists h \in \mathcal{H} : R_{Emp}(h) = 0 \text{ et } R_{Réel}(h) > \varepsilon \} \quad (17.1)$$

Une idée essentielle consiste à chercher une *mesure effective* de la variabilité de \mathcal{H} en la mettant à l'épreuve sur un ensemble de points test tirés aléatoirement. Le risque réel sera alors évalué en utilisant un autre échantillon que l'échantillon d'apprentissage, qui sera lui aussi issu d'un tirage i.i.d. (formes tirées aléatoirement dans \mathcal{X} suivant une même distribution). Nous le nommerons naturellement *échantillon de test*, noté \mathcal{T} . L'avantage est que l'on pourra alors *compter* les différents étiquetages de ces points que permet l'espace des fonctions hypothèses \mathcal{H} , et à partir de là caractériser la variabilité effective de \mathcal{H} .

On voudrait donc remplacer l'étude de l'expression (17.1) par celle de:

$$P_{\mathcal{D}_{\mathcal{Z}}} \{ \mathcal{S} : \exists h \in \mathcal{H} : R_{Emp}^{\mathcal{S}}(h) = 0 \text{ et } R_{Emp}^{\mathcal{T}}(h) = 0 > \varepsilon \} \quad (17.2)$$

où nous notons $R_{Emp}^{\mathcal{S}}(h)$ (resp. $R_{Emp}^{\mathcal{T}}(h)$) le risque empirique de l'hypothèse h mesuré sur l'échantillon \mathcal{S} (resp. \mathcal{T}).

Une application des inégalités de Chernoff¹ avec $m > 2/\varepsilon$, permet de borner l'expression (17.1) par une probabilité relative à l'expression (17.2). Plus précisément, si on note \mathcal{ST} l'échantillon constitué de la concaténation de \mathcal{S} et de \mathcal{T} , et $R_{Emp}^{\mathcal{ST}}(h)$ le risque empirique de h mesuré sur \mathcal{ST} , on peut obtenir la borne :

$$\begin{aligned} P_{\mathcal{D}_{\mathcal{X}}} \{ \mathcal{S} : \exists h \in \mathcal{H} : R_{Emp}^{\mathcal{S}}(h) = 0 \text{ et } R_{Réel}(h) > \varepsilon \} \\ \leq 2P_{\mathcal{D}_{\mathcal{X}}} \{ \mathcal{ST} : \exists h \in \mathcal{H} : R_{Emp}^{\mathcal{S}}(h) = 0 \text{ et } R_{Emp}^{\mathcal{ST}}(h) > \frac{\varepsilon}{2} \} \end{aligned} \quad (17.3)$$

Répétons que cette nouvelle borne représente un progrès considérable dans la solution du problème initial. En effet, nous sommes maintenant ramenés à un comptage sur les étiquettes

1. Pour les détails de la démonstration assez longue dont les développements dépassent le cadre de notre ouvrage, il est suggéré de se reporter à [AB92] pp.90-94 ou à [KV94] pp.59-61, ou encore à [AB96] pp.42-50. [DGL96] chap.12 offre aussi une exposition très complète de la question.

possibles de $2m$ points par les fonctions de \mathcal{H} , au lieu d'avoir à considérer l'ensemble infini des fonctions de \mathcal{H} .

Soit un échantillon i.i.d. suivant $\mathcal{D}_{\mathcal{X}}$ de $2m$ points, dont $l \geq \varepsilon m/2$ sont mal étiquetés par l'hypothèse h . Le nombre de cas possibles pour lesquels aucune de ces erreurs n'intervient dans les m premiers points est : C_m^l/C_{2m}^l . Or : $C_m^l/C_{2m}^l \leq 1/2^l$, puisque :

$$\frac{C_m^l}{C_{2m}^l} = \prod_{i=0}^{i=l-1} \frac{m-i}{2m-i} \leq \prod_{i=0}^{i=l-1} \left(\frac{1}{2}\right) = \frac{1}{2^l}$$

La probabilité que cela arrive pour n'importe quel échantillon de taille $2m$ et pour n'importe quelle hypothèse est bornée par le nombre d'étiquetages différents de $2m$ points par des fonctions de \mathcal{H} . En d'autres termes, on évalue maintenant la richesse de l'espace d'hypothèses par le nombre de *dichotomies* différentes qu'il peut induire au maximum sur un échantillon aléatoire de $2m$ points. On appelle ce nombre la *fonction de croissance* de \mathcal{H} : $G_{\mathcal{H}}(2m)$.

$$P_{\mathcal{D}_{\mathcal{X}}} \{ \mathcal{S} : \exists h \in \mathcal{H} : R_{Emp}^{\mathcal{S}}(h) = 0 \text{ et } R_{Réel}(h) > \varepsilon \} \leq 2 G_{\mathcal{H}}(2m) 2^{-\varepsilon m/2} \quad (17.4)$$

Cette équation est très importante. Elle nous montre que la probabilité qu'il existe une mauvaise hypothèse (pour laquelle le risque empirique nul est trompeur) est bornée par un produit de deux termes, dont l'un est une exponentielle décroissante en m .

Il nous faut donc étudier le second terme : la fonction de croissance. En effet, l'inégalité ci-dessus n'est **utile que si la fonction de croissance croît à un rythme sub-exponentiel** pour laquelle la borne tend vers zéro rapidement.

17.1.2 Fonction de croissance et dimension de Vapnik-Chervonenkis

Définition 17.1 (Fonction de croissance)

La fonction de croissance $G_{\mathcal{H}}(m)$ définit le nombre maximal de dichotomies qui peuvent être induites par l'espace des fonctions indicatrices \mathcal{H} sur un échantillon de taille m sur \mathcal{X} .

Il est important de noter que cette définition invoque l'échantillon le plus grand pour lequel il est possible de réaliser toutes les dichotomies (et non pas n'importe quel échantillon tiré aléatoirement).

Cette fonction est indépendante de toute distribution de probabilité sur \mathcal{X} et ne dépend que de l'espace \mathcal{H} . Pour m points, le nombre maximal possible de dichotomies, c'est-à-dire d'étiquetages possibles, est de 2^m . Nous avons donc : $G_{\mathcal{H}}(m) \leq 2^m$.

On dit d'un ensemble de points $\{\mathbf{x}_1, \dots, \mathbf{x}_m\}$ pour lequel \mathcal{H} permet d'induire les 2^m dichotomies possibles qu'il est *pulvérisé(shattered set)*. Dans ce cas, $G_{\mathcal{H}}(m) = 2^m$, c'est-à-dire que \mathcal{H} permet de réaliser n'importe quelle fonction indicatrice de ces m points.

Exemple 12 (Séparatrices linéaires dans le plan)

La figure 17.1 illustre un espace d'entrée \mathcal{X} de dimension 2 avec ici 4 points : $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4$. Les bornes des fonctions h_1 et h_2 sont indiquées par des ellipses. D'après cette figure, les fonctions h_1 et h_2 induisent respectivement les dichotomies :

$$\mathcal{D}_1 = \{\mathcal{S}_- = \{\mathbf{x}_3\}, \mathcal{S}_+ = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_4\}\} \quad \text{et} \quad \mathcal{D}_2 = \{\mathcal{S}_- = \{\mathbf{x}_1, \mathbf{x}_2\}, \mathcal{S}_+ = \{\mathbf{x}_3, \mathbf{x}_4\}\}$$

L'ensemble des dichotomies possibles de l'ensemble \mathcal{S} de ces 4 points est : $|\Delta_{\mathcal{S}}| = 2^4 = 16$.

Pour la classe \mathcal{H} des concepts définis par les séparatrices linéaires, il est aisément de voir que tout ensemble de trois points non colinéaires peut être pulvérisé. La figure 17.2(a) montre l'une des

huit dichotomies possibles de trois points du plan. Le lecteur pourra facilement vérifier que les sept autres dichotomies sont également réalisables par une hypothèse de \mathcal{H} . Nous avons donc : $G_{\mathcal{H}}(3) = 2^3$.

Nous montrons qu'il est impossible de pulvériser un ensemble de quatre points quelconques par des séparatrices linéaires en considérant les deux cas génériques (l'un où tous les points sont sur l'enveloppe convexe, l'autre où seuls trois points définissent l'enveloppe convexe) (figure 17.2(b) et (c)) pour lesquels il n'existe pas de séparatrice linéaire permettant d'induire les dichotomies correspondantes.

Nous avons donc : $G_{\mathcal{H}}(4) \leq 2^4$

□

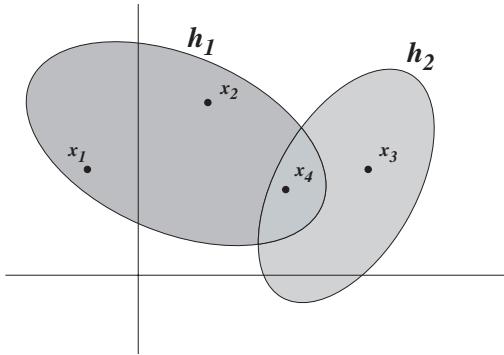


FIG. 17.1 – Un espace d'entrée \mathcal{X} de dimension 2 avec les dichotomies induites par deux fonctions h_1 et h_2 .

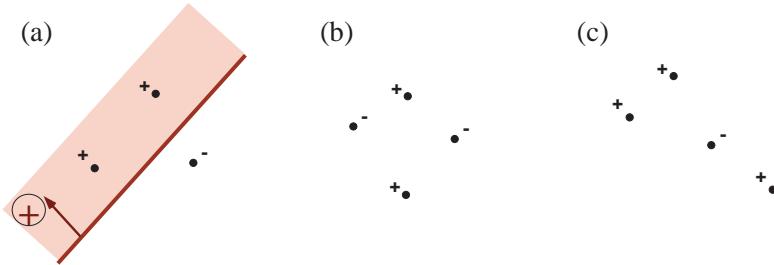


FIG. 17.2 – (a) Une dichotomie et une réalisation par une séparatrice linéaire. (b) et (c) Dichotomies irréalisables par des séparatrices linéaires.

Exemple 13 (Rectangles parallèles aux axes dans le plan)

Il est également possible de réaliser toutes les dichotomies possibles de trois points du plan par des rectangles parallèles aux axes. Seul le cas dégénéré où le point négatif est exactement au milieu des points positifs rend impossible cette dichotomie (voir la figure 17.3 (a)) pour un exemple). Par ailleurs, il existe un échantillon de quatre points pour lesquels il est possible de réaliser tous les étiquetages possibles à l'aide de rectangles parallèles aux axes (voir la figure 17.3(a), nous laissons au lecteur le soin de le vérifier pour les quinze autres étiquetages). Cependant, il n'est pas possible de former toutes les dichotomies de quatre points quelconques, comme l'illustre la figure 17.3 (b). L'existence d'un seul échantillon de quatre points pouvant être pulvérisé suffit à borner la fonction de croissance par le bas $G_{\mathcal{H}}(4) = 2^4$. Si nous considérons maintenant n'importe quel échantillon de cinq points, l'un de ces points est nécessairement ni en position extrême gauche, ni en position extrême droite, ni en position extrême haute, ni en position

extrême basse (voir figure 17.3 (d)). Si nous étiquetons ce point non extrémal par un '−', et les quatre autres points par '+', alors il est impossible de trouver un rectangle permettant de couvrir ces 4 points sans couvrir le cinquième. D'où $G_{\mathcal{H}}(5) = 2^4$. \square

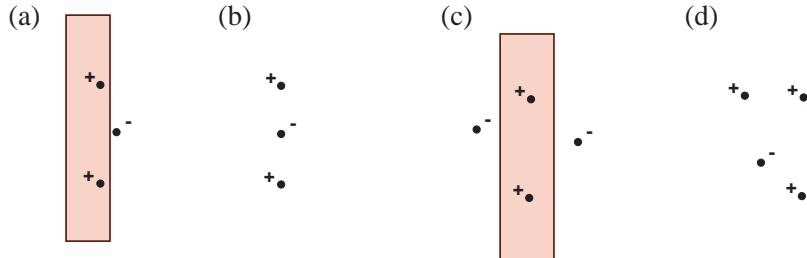


FIG. 17.3 – (a) Exemple de trois points dont la dichotomie est réalisable par au moins un rectangle. (b) Un exemple de trois points ne pouvant être distingués grâce à un rectangle. Mais il s'agit du seul cas, correspondant à un cas dégénéré du cas général à trois points (a). (c) Une dichotomie de quatre points et une réalisation par un rectangle parallèle aux axes. (d) Une dichotomie irréalisable.

Définition 17.1

On appelle **dimension de Vapnik-Chervonenkis** (ou dimension entière) d'un espace de fonctions binaires \mathcal{H} que l'on note $d_{VC}(\mathcal{H})$ le cardinal du plus grand ensemble de points de l'espace d'entrée \mathcal{X} qu'il est possible de pulvériser. Si il n'y a pas de maximum, par convention $d_{VC}(\mathcal{H}) = \infty$.

En d'autres termes, la dimension de Vapnik-Chervonenkis d'un ensemble de fonctions de discriminations est le nombre maximal d'exemples d'apprentissage dont il est possible d'apprendre n'importe quel étiquetage. Cela signifie qu'il existe un ensemble de points au moins dont le cardinal est égal à $d_{VC}(\mathcal{H})$ et sur lequel la « souplesse » de \mathcal{H} est totale.

La dimension de Vapnik-Chervonenkis est donc par définition liée à la fonction de croissance par la relation :

$$d_{VC}(\mathcal{H}) = \max \{m \in \mathbb{N} : G_{\mathcal{H}}(m) = 2^m\} \quad (17.5)$$

Exemple 14 (Séparatrices linéaires dans le plan)

D'après l'exemple 12, la dimension de Vapnik-Chervonenkis pour les fonctions séparatrices linéaires du plan est égale à 3.

Le lecteur pourra démontrer que la dimension de Vapnik-Chervonenkis des séparatrices linéaires sur \mathbb{R}^d est égale à $d + 1$. \square

Il nous reste à voir en quoi la dimension de Vapnik-Chervonenkis est plus précisément reliée à la fonction de croissance pour nous permettre de trouver les conditions de croissance subexponentielle de cette dernière lesquelles, rappelons-le, sont nécessaires pour garantir l'utilité du principe inductif *ERM*.

17.1.3 Le lemme de Sauer : un lemme sauveur

Nous avons vu que la fonction de croissance $G_{\mathcal{H}}(m)$ est bornée par 2^m . Mais est-ce que toute valeur est possible en dessous de cette limite ? (Voir figure 17.4). Le lemme démontré

indépendamment par Sauer (1972), Shelah (1972) et Vapnik et Chervonenkis (1971) montre que ce n'est pas le cas si la dimension de Vapnik-Chervonenkis est finie.

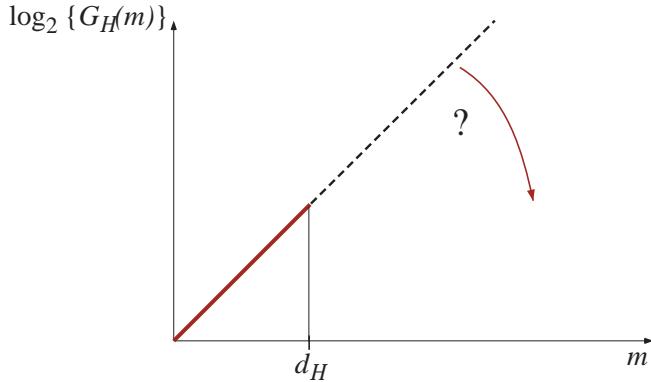


FIG. 17.4 – *Le comportement de la fonction de croissance. Nous avons tracé le logarithme en base 2 de la fonction de croissance en fonction du nombre de points dans l'espace d'entrée \mathcal{X} . Jusqu'à $d_{\mathcal{H}}$, la dimension de Vapnik-Chervonenkis de \mathcal{H} , $G_{\mathcal{H}}(m) = 2^m$, d'où le tracé linéaire. Mais pour des valeurs de $m \geq d_{\mathcal{H}}$, quelle est l'enveloppe maximale des dichotomies possibles de m points dans \mathcal{X} par des fonctions de \mathcal{H} ?*

Théorème 17.1 (Lemme de Sauer)

Soient \mathcal{H} un espace de fonctions hypothèses indicatrices sur l'espace d'entrée \mathcal{X} , $d_{\mathcal{H}}$ sa dimension de Vapnik-Chervonenkis : $d_{\mathcal{H}} = d_{VC}(\mathcal{H})$, et m points quelconques dans \mathcal{X} . Alors² :

$$G_{\mathcal{H}}(m) \leq \sum_{i=0}^{d_{\mathcal{H}}} C_m^i \leq \left(\frac{em}{d_{\mathcal{H}}}\right)^{d_{\mathcal{H}}} \in \mathcal{O}(m^{d_{\mathcal{H}}}) \quad (17.6)$$

Démonstration. Nous allons d'abord borner le nombre de dichotomies réalisables sur un échantillon \mathcal{S} de m points quand $d_{\mathcal{H}} = d < m$ par une somme de combinaisons, puis nous montrerons que cette somme est bornée par une fonction polynomiale de m . C'est la première partie de la démonstration qui est la plus importante, et la plus intéressante.

Supposons par hypothèse que la fonction de croissance $G_{\mathcal{H}}(m)$ soit bornée par une fonction $\phi_{d_{\mathcal{H}}}(m)$. Soit $\pi_{\mathcal{H}}(\mathcal{S})$ l'ensemble des dichotomies de \mathcal{S} réalisables à l'aide des fonctions de \mathcal{H} , et soit \mathbf{x} un point de \mathcal{S} . Nous voudrions calculer $|\pi_{d_{\mathcal{H}}}(\mathcal{S})|$ à partir de $|\pi_{d_{\mathcal{H}}}(\mathcal{S} - \mathbf{x})|$, et ainsi obtenir une expression récursive de $|\pi_{d_{\mathcal{H}}}(\mathcal{S})|$.

La différence entre $|\pi_{d_{\mathcal{H}}}(\mathcal{S})|$ et $|\pi_{d_{\mathcal{H}}}(\mathcal{S} - \mathbf{x})|$ provient des ensembles de dichotomies de \mathcal{S} qui ne diffèrent que par l'étiquette de \mathbf{x} , mais qui sont donc non distinguées dans $\pi_{\mathcal{H}}(\mathcal{S} - \mathbf{x})$. Elles correspondent à un ensemble $D_{\mathcal{S}-\mathbf{x}}$ de dichotomies dans $\pi_{\mathcal{H}}(\mathcal{S} - \mathbf{x})$. Nous avons :

$$|\pi_{d_{\mathcal{H}}}(\mathcal{S})| = |\pi_{d_{\mathcal{H}}}(\mathcal{S} - \mathbf{x})| + |D_{\mathcal{S}-\mathbf{x}}|$$

Il faut donc compter le nombre de ces paires de dichotomies de $D_{\mathcal{S}-\mathbf{x}}$. Quelle est la dimension de Vapnik-Chervonenkis de cet ensemble ?

Il est facile de voir qu'elle est nécessairement strictement inférieure à $d_{\mathcal{H}}$. En effet, si elle était égale à $d_{\mathcal{H}}$ (si toutes les dichotomies de $\pi_{\mathcal{H}}(\mathcal{S}-\mathbf{x})$ étaient réalisables), alors toutes les dichotomies

2. Pour une démonstration, on peut se reporter à [AB92] pp.79-83 ou à [KV94] pp.54-57.

de \mathcal{S} le seraient aussi, puisque pour chaque dichotomie de $\pi_{\mathcal{H}}(\mathcal{S} - \mathbf{x})$ on peut étiqueter \mathbf{x} de deux manières différentes. Et donc on aurait $d_{\mathcal{H}} = d + 1$, ce qui contredirait notre hypothèse de départ.

Nous avons donc : $|\phi_{d_{\mathcal{H}}}(\mathcal{S})| = |\phi_{d_{\mathcal{H}}}(\mathcal{S} - \mathbf{x})| + |D_{\mathcal{S}-\mathbf{x}}|$, soit encore :

$$\phi_d(m) = \phi_d(m-1) + \phi_{d-1}(m-1)$$

avec $\phi_d(0) = \phi_0(m) = 1$ (ce qui est évident).

Il est alors facile de montrer que : $\phi_d(m) = \sum_{i=0}^{d_{\mathcal{H}}} C_m^i$.

Le deuxième temps de la démonstration est simplement technique.

Pour $0 \leq i \leq d$ et $m \geq d$: $(m/d)^d (d/m)^i \geq 1$, d'où :

$$\sum_{i=0}^{d_{\mathcal{H}}} C_m^i \leq (m/d)^d \sum_{i=0}^{d_{\mathcal{H}}} C_m^i (d/m)^i \leq (m/d)^d (1 + d/m)^m < \left(\frac{em}{d_{\mathcal{H}}}\right)^{d_{\mathcal{H}}}$$

en utilisant des développements en série limitée classiques. \square

Le comportement de la fonction de croissance $G_{\mathcal{H}}(m)$ est donc contrôlé par la dimension de Vapnik-Chervonenkis $d_{\mathcal{H}}$. De manière *a priori* surprenante, pour $m > d_{\mathcal{H}}$, il n'y a donc pas de fonction de croissance possible au-dessus de la borne polynomiale $(em/d_{\mathcal{H}})^{d_{\mathcal{H}}}$. Par exemple, il n'est pas possible d'avoir une fonction de croissance en \sqrt{m} . **La dimension de Vapnik-Chervonenkis correspond à une borne fondamentale de la capacité d'expression d'un ensemble de fonctions hypothèses.**

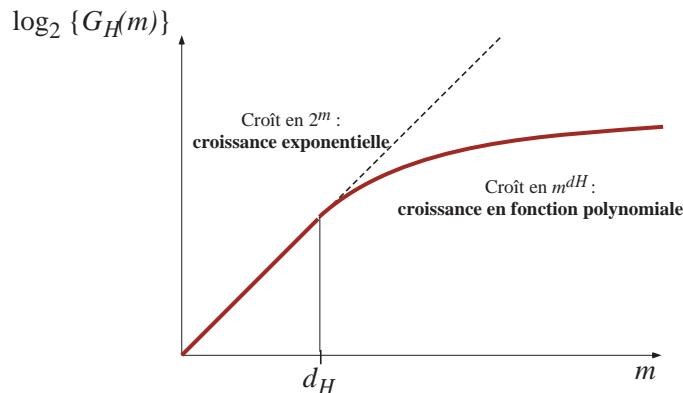


FIG. 17.5 – Le comportement de la fonction de croissance d'après le lemme de Sauer. On a tracé ici le logarithme en base 2 de la fonction de croissance.

La figure 17.5 illustre le comportement de la fonction de croissance d'après le lemme de Sauer. Il est intéressant d'en prendre le logarithme Népérien :

$$\ln \{G_{\mathcal{H}}(m)\} = \ln \left(\frac{em}{d_{\mathcal{H}}} \right)^{d_{\mathcal{H}}} = d_{\mathcal{H}} \left(1 + \ln \frac{m}{d_{\mathcal{H}}} \right) \quad (17.7)$$

En reprenant l'équation 17.4, nous arrivons à l'équation fondamentale :

$$\begin{aligned} P_{\mathcal{D}_{\mathcal{X}}} \{ \mathcal{S} : \exists h \in \mathcal{H} : R_{Emp}^{\mathcal{S}}(h) = 0 \text{ & } R_{Réel}(h) > \varepsilon \} &\leq 2 G_{\mathcal{H}}(2m) 2^{-\varepsilon m/2} \\ &\leq 2 \left(\frac{2em}{d_{\mathcal{H}}} \right)^{d_{\mathcal{H}}} 2^{-\varepsilon m/2} \end{aligned} \quad (17.8)$$

Cette équation montre que si la dimension de Vapnik-Chervonenkis $d_{\mathcal{H}}$ est finie, alors la convergence du risque empirique vers le risque réel est exponentiellement rapide, d'après le terme en $2^{-\varepsilon m/2}$, et ceci uniformément pour n'importe quelle fonction $h \in \mathcal{H}$.

À partir de cette équation, on peut aussi calculer la taille minimale d'un échantillon d'apprentissage pour que la probabilité dénotée par l'équation 17.8 soit inférieure à δ :

$$m = \mathcal{O}\left(\frac{1}{\varepsilon} \ln \frac{1}{\delta} + \frac{d_{\mathcal{H}}}{\varepsilon} \ln \frac{1}{\varepsilon}\right) \quad (17.9)$$

Exemple 15 (Application numérique)

Soit $d_{\mathcal{H}} = 3$, la dimension de Vapnik-Chervonenkis de l'espace d'hypothèses (il s'agit par exemple de l'espace des séparatrices linéaires du plan, telles qu'un perceptron pourrait les réaliser). On veut une erreur d'approximation limitée à $\varepsilon = 1\%$ pour un seuil de confiance de 95 % ($\delta = 5\%$). Alors, il faut avoir un échantillon d'apprentissage de taille $m \geq 2425$. \square

Voyons maintenant comment traduire l'équation 17.8 en une borne d'erreur pour toute hypothèse h consistante.

Théorème 17.2 (Théorème PAC pour les espaces infinis de fonctions indicatrices)

Soit \mathcal{H} un espace de fonctions indicatrices de dimension de Vapnik-Chervonenkis $d_{\mathcal{H}}$. Pour toute distribution de probabilité \mathcal{D} sur $\mathcal{X} \times \{-1, 1\}$, avec une probabilité $1 - \delta$ sur les échantillons \mathcal{S} de m exemples tirés aléatoirement sur \mathcal{Z} , toute hypothèse $h \in \mathcal{H}$ cohérente avec l'échantillon \mathcal{S} est d'erreur réelle bornée par :

$$R_{R\acute{e}el}(h) \leq \frac{2}{m} \left(d_{\mathcal{H}} \ln \frac{2em}{d_{\mathcal{H}}} + \ln \frac{2}{\delta} \right) \quad (17.10)$$

pourvu que $d_{\mathcal{H}} \leq m$ et $m > 2/\varepsilon$.

Ce théorème montre que la taille de l'échantillon d'apprentissage requise pour assurer (en probabilité) une bonne généralisation est une **fonction linéaire de la dimension de Vapnik-Chervonenkis de l'espace d'hypothèses** quand on choisit une hypothèse cohérente, et ceci face à toute distribution des exemples.

17.1.4 L'analyse de Vapnik et Chervonenkis pour des fonctions quelconques

L'analyse précédente est limitée sur un certain nombre de points. Elle ne s'applique en effet :

- qu'à des fonctions indicatrices, donc à des applications d'apprentissage de fonctions de discrimination (apprentissage de concepts) ;
- qu'à des fonctions de perte comptant le nombre d'erreurs de classification (fonction de perte pour la discrimination) ;
- qu'au cas où l'espace d'hypothèses \mathcal{H} contient l'espace de fonctions cible \mathcal{F} , ce qui permet d'assurer l'existence d'au moins une hypothèse cohérente avec l'échantillon d'apprentissage.

Les travaux de Vapnik et Chervonenkis, menées avec opiniâtreté sur une longue période (1968-1998 environ), ont permis de lever ces limitations et de fournir une approche générale du problème de l'induction pour des espaces infinis de fonctions hypothèse quelconques (ou presque) utilisant des fonctions de pertes quelconques (ou presque), y compris dans le cas où les espaces \mathcal{H} et \mathcal{F} ne coïncident pas. Cette analyse sert actuellement de cadre de référence pour toute la théorie de l'apprentissage.

Nous en donnons un rapide aperçu dans ce qui suit, reportant le lecteur intéressé aux nombreuses publications concernant ce sujet, et en particulier à [Vap95] pour un petit livre d'un abord facile résumant l'approche, et à [Vap98] pour une description détaillée. Le lecteur motivé pourra également se reporter avec profit à [AB96] qui donne de nombreuses preuves. [DGL96] est également une référence intéressante, qui ne traite toutefois que des problèmes de classification binaire.

17.1.4.1 Le théorème fondamental de la théorie de l'apprentissage selon Vapnik

Rappelons que selon Vapnik, le problème central de l'induction est de chercher quelle relation existe entre une hypothèse sélectionnée selon le principe inductif *ERM*, qui minimise le risque empirique, et l'hypothèse optimisant le risque réel. Il s'agit en particulier d'étudier les conditions de pertinence du principe *ERM* correspondant aux équations (2.10) et (2.12). Vapnik et Chervonenkis [Vap82, VC91] ont montré le théorème suivant.

Théorème 17.3 (Théorème fondamental de pertinence de l'*ERM* (Vapnik))

*Pour des fonctions de perte bornées, le principe de minimisation du risque empirique (*ERM*) est pertinent si et seulement si le risque empirique converge uniformément vers le risque réel au sens suivant :*

$$\lim_{m \rightarrow \infty} P \left[\sup_{\hat{h}_{\mathcal{S}} \in \mathcal{H}} |R_{Réel}(\hat{h}_{\mathcal{S}}) - R_{Emp}(\hat{h}_{\mathcal{S}})| > \varepsilon \right] = 0, \quad \forall \varepsilon > 0 \quad (17.11)$$

À nouveau, insistons sur ce point : ce théorème³ signifie quelque chose de très profond et de très important : la pertinence du principe de minimisation du risque empirique est déterminée par la *pire* des fonctions hypothèse de \mathcal{H} , c'est-à-dire celle dont l'écart entre le risque empirique mesuré et le risque réel est le plus grand. C'est là l'essence des convergences uniformes. Nous nous trouvons ici dans une théorie qui spécifie des garanties de généralisation face à n'importe quelle distribution des exemples tirés dans l'échantillon d'apprentissage (d'où la convergence en probabilité et le seuil de confiance δ) et face à la pire des fonctions hypothèse possibles, au sens où, choisie selon le principe inductif *ERM*, elle se révélerait de fait la moins bonne.

Un théorème de convergence uniforme nécessite une propriété sur l'ensemble \mathcal{H} des fonctions hypothèse. Pour assurer la pertinence du principe *ERM* par l'équation (17.11), il faut donc se donner une *mesure de diversité* sur \mathcal{H} . L'une de ces mesures de diversité ou de richesse de l'espace d'hypothèses est évidemment la *dimension de Vapnik-Chervonenkis*. C'est elle qui a permis à Vapnik et Chervonenkis d'énoncer une loi des grands nombres pour les espaces fonctionnels.

Théorème 17.4 (Conditions nécessaires et suffisantes pour la pertinence de l'*ERM*)

*L'équation suivante fournit une condition nécessaire et suffisante pour la pertinence du principe *ERM*, ainsi qu'une garantie de convergence rapide, indépendamment de la distribution de probabilité sur les échantillons \mathcal{S} .*

$$\lim_{m \rightarrow \infty} \frac{\ln G_{\mathcal{H}}(m)}{m} = 0 \quad (17.12)$$

Nous retrouvons ici la nécessité, soulignée dans l'analyse *PAC*, que la fonction de croissance croisse moins vite qu'une fonction exponentielle de la taille m de l'échantillon.

3. Pour une preuve partielle, se reporter par exemple à [Hay99] p.93.

17.1.4.2 Bornes utiles sur la capacité de généralisation

En reprenant la figure 17.6, les questions centrales dans l'analyse du principe inductif *ERM* et de sa consistance sont :

1. Quel est le risque réel associé à l'hypothèse \hat{h}_S qui minimise le risque empirique, et quelle est la relation liant $R_{Réel}(\hat{h}_S)$ et $R_{Emp}(\hat{h}_S)$?
2. Quelle est la proximité entre le risque réel $R_{Réel}(\hat{h}_S)$ associé à l'hypothèse \hat{h}_S sélectionnée par le principe inductif *ERM* et le risque réel optimal $R_{Réel}(h^*)$?

Les réponses à ces questions peuvent être obtenues par le calcul de bornes de taux de convergence découlant de l'analyse théorique conduite par Vapnik et Chervonenkis. Ces bornes sont fonctions de la taille m de l'échantillon d'apprentissage, des propriétés de l'espace de fonctions hypothèse \mathcal{H} et de celles des fonctions de perte $l(u_i, h(x_i))$. Nous nous limitons ici à la présentation de résultats concernant les fonctions de perte positives bornées (correspondant aux problèmes de classification). Des bornes pour d'autres fonctions de perte sont discutées notamment dans [Vap95] et [Vap98].

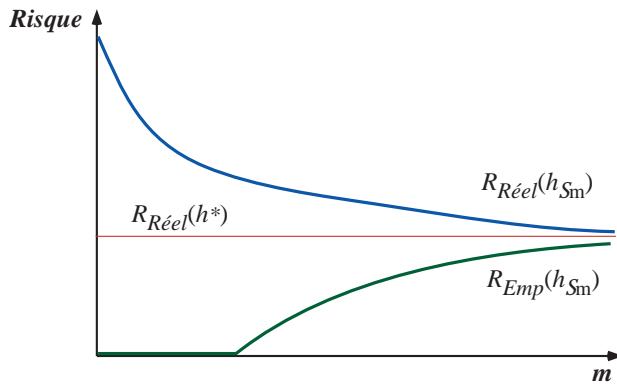


FIG. 17.6 – *Reprise de la figure 2.11.*

Les fonctions de perte positives bornées. On suppose que la fonction de perte $l(u_i, h(x_i))$ est bornée par B quelle que soit la fonction hypothèse h , les exemples x_i et les réponses désirées u_i .

Alors les réponses aux questions ci-dessus sont fournies par les bornes suivantes :

1. L'inégalité suivante est vérifiée avec une probabilité $\geq 1 - \delta$ simultanément pour toutes les fonctions de \mathcal{H} , et en particulier la fonction \hat{h}_S minimisant le risque empirique (condition de convergence uniforme) :

$$R_{Réel}(h) \leq R_{Emp}(h) + \frac{B\epsilon}{2} \left(1 + \sqrt{1 + \frac{4R_{Emp}(h)}{B\epsilon}} \right) \quad (17.13)$$

où :

$$\epsilon = 4 \frac{G_{\mathcal{H}}(2m) - \ln \delta / 4}{m} = 4 \frac{d_{\mathcal{H}}[\ln \frac{2m}{d_{\mathcal{H}}} + 1] - \ln \delta / 4}{m}$$

2. L'inégalité suivante est valide avec une probabilité $\geq 1 - 2\delta$ pour la fonction \hat{h}_S qui minimise le risque empirique :

$$R_{Réel}(\hat{h}_S) - R_{Réel}(h^*) \leq B \sqrt{\frac{-\ln \delta}{2m}} + \frac{B\epsilon}{2} \left(1 + \sqrt{1 + \frac{4}{\epsilon}} \right) \quad (17.14)$$

Il faut noter que la fonction de perte classique en classification prenant ses valeurs dans $\{0, 1\}$, la borne B est égale à 1 dans ce cas. Par ailleurs, dans le cas de l'analyse *PAC* où le risque empirique optimal est nul, on obtient pour l'équation 17.13, l'expression suivante:

$$R_{Réel}(h) \leq R_{Emp}(h) + 2 \frac{d_{\mathcal{H}}[\ln \frac{2m}{d_{\mathcal{H}}} + 1] - \ln \delta/4}{m} \quad (17.15)$$

Il ressort de ces inégalités le **message important** que si la dimension de Vapnik-Chervonenkis de l'espace d'hypothèses est finie, alors l'erreur d'estimation entre $R_{Réel}(\hat{h}_{\mathcal{S}})$ et $R_{Réel}(h^*)$ converge vers zéro au rythme de $O(\sqrt{\frac{d_{\mathcal{H}} \ln m}{m}})$ dans le cas général et de $O(\frac{d_{\mathcal{H}} \ln m}{m})$ dans le cas où \mathcal{H} comprend la fonction cible f^* , et ceci *pour toute distribution* sur $\mathcal{X} \times \mathcal{U}$.

La beauté de ce résultat et sa puissance proviennent de sa nature « pour-toute-distribution » et du fait que les propriétés de la classe \mathcal{H} des fonctions hypothèses sont reflétées à travers un seul paramètre : sa dimension de Vapnik-Chervonenkis.

Il est important de souligner, que d'après [Vap95] en p.81, ces bornes ne peuvent pas être significativement améliorées. Ce sont des bornes supérieures assez serrées dans la mesure où les bornes inférieures connues sont du même ordre de grandeur (de l'ordre de $\sqrt{d_{\mathcal{H}}/m}$ au lieu de $\sqrt{(d_{\mathcal{H}}/m) \ln(d_{\mathcal{H}}/m)}$) et donc les bornes supérieures ci-dessus sont optimales à un facteur logarithmique près, et sont donc d'importance pratique aussi bien que théorique.

Remarque

Afin de pouvoir appliquer les bornes de la théorie à des problèmes pratiques, il faut pouvoir estimer de manière précise la dimension de Vapnik-Chervonenkis de l'espace d'hypothèses concerné. Malheureusement, l'estimation analytique de la dimension de Vapnik-Chervonenkis n'est possible que pour des classes de fonctions simples. Il faut donc souvent avoir recours à des *estimations heuristiques* de $d_{\mathcal{H}}$, par exemple en observant le risque empirique sur des échantillons de données indépendants et de tailles différentes, comme le propose Vapnik dans [VLC94]. Le problème de l'évaluation empirique de la dimension de Vapnik-Chervonenkis est un sujet de recherche important.

Remarque

Ce chapitre a essentiellement pour but d'introduire le lecteur à un certain nombre de questions fondamentales sur l'induction. Nous avons à cet effet laissé sous silence certaines subtilités. C'est ainsi que par exemple, nous avons supposé que la tâche de classification binaire était réalisée par un apprenant implémentant des fonctions indicatrices. Lorsque, pour cette même tâche, l'apprenant implémente des fonctions à valeur réelle, alors de nouvelles analyses plus fines du principe *ERM* peuvent être conduites. Ce sont en particulier les analyses portant sur les *classificateurs à large marge* (*large margin classifiers*). Une réalisation concerne les machines à vastes marges décrites dans le chapitre 9.

17.1.5 Discussion

Finalement, le principe inductif de minimisation du risque empirique (*ERM*) est-il légitime ? Les analyses théoriques (*PAC* pour les fonctions indicatrices et de Vapnik et Chervonenkis pour les fonctions quelconques) montrent que cela dépend de la taille de l'échantillon d'apprentissage et plus précisément du rapport $m/d_{\mathcal{H}}$ qui prend en compte la dimension de Vapnik-Chervonenkis de l'espace des fonctions hypothèse \mathcal{H} . Si en effet nous considérons les inégalités (17.13) et (17.15), nous voyons apparaître deux cas :

- $m/d_{\mathcal{H}}$ est **grand**. Dans ce cas, ϵ est petit, et le second terme des inégalités (17.13) et (17.15) devient petit. Le risque réel est alors proche du risque empirique, et dans ce cas une valeur faible du risque empirique garantit (en probabilité) une valeur faible du risque réel.

Le principe *ERM* est alors justifié.

- **$m/d_{\mathcal{H}}$ est petit.** Dans ce cas, une faible valeur du risque empirique ne garantit rien sur la valeur du risque réel. Pour minimiser le risque réel, il faut aussi minimiser le terme de droite des inégalités (17.13) et (17.15) en prenant en compte les deux termes simultanément. Or le premier terme dépend d'une fonction spécifique dans \mathcal{H} , tandis que le second terme dépend de l'ensemble des fonctions \mathcal{H} par l'intermédiaire de sa dimension de Vapnik-Chervonenkis. Pour minimiser le terme de droite de ces inégalités, il faut donc maintenant faire de $d_{\mathcal{H}}$ un paramètre de contrôle. Nous verrons cette idée à l'œuvre dans un nouveau principe inductif proposé par Vapnik : le *principe de minimisation du risque structurel (SRM: Structural Risk Minimization)* (Voir la section 17.2.1).

17.2 Les principes inductifs avec contrôle de l'espace des hypothèses

17.2.1 La minimisation du risque structurel : SRM

Au terme de l'étude sur la cohérence du principe de minimisation du risque empirique, c'est-à-dire du lien entre le risque empirique minimal, le risque réel associé et le risque réel optimal, Vapnik et ses collègues ont obtenu des bornes, valables pour toute fonction cible et pour toute distribution des exemples, sous la forme générale (pour le cas du problème de classification) :

$$R_{Réel}(\hat{h}_{\mathcal{S}}^d) \leq R_{Emp}(\hat{h}_{\mathcal{S}}^d) + \Phi\left(\frac{m}{d_{\mathcal{H}_d}}\right) \quad (17.16)$$

où $d_{\mathcal{H}_d}$, la dimension de Vapnik-Chervonenkis, mesure la capacité de l'espace d'hypothèses \mathcal{H}_d . $\hat{h}_{\mathcal{S}}^d$ est l'hypothèse de risque empirique minimal dans l'espace \mathcal{H}_d et Φ est une fonction de la taille de l'échantillon d'apprentissage m et de la dimension de Vapnik-Chervonenkis qui mesure un intervalle de confiance. On a une équation générale similaire pour le cas de la régression :

$$R_{Réel}(\hat{h}_{\mathcal{S}}^d) \leq \frac{R_{Emp}(\hat{h}_{\mathcal{S}}^d)}{\Phi'\left(\frac{m}{d_{\mathcal{H}_d}}\right)} \quad (17.17)$$

Sur cette base, Vapnik propose un nouveau principe inductif baptisé *principe de minimisation du risque structurel (Structural Risk Minimization : SRM)* reposant sur deux idées.

1. La capacité d'une classe d'hypothèses \mathcal{H} va être mesurée par sa dimension de Vapnik-Chervonenkis $d_{\mathcal{H}}$. Il est alors possible de définir une *structure* sur les classes d'hypothèses⁴ consistant en une séquence enchaînée de classes d'hypothèses \mathcal{H}_d : $\mathcal{H}_1 \subseteq \mathcal{H}_2 \subseteq \dots \subseteq \mathcal{H}_d \subseteq \dots$ telle que chaque classe \mathcal{H}_d est de dimension de Vapnik-Chervonenkis $d_{\mathcal{H}_d}$ finie avec : $d_{\mathcal{H}_1} \leq d_{\mathcal{H}_2} \leq \dots \leq d_{\mathcal{H}_d} \leq \dots$
 2. Le choix, par *ERM*, de la meilleure hypothèse $\hat{h}_{\mathcal{S}}^d$, et donc du meilleur espace d'hypothèses, se fait en sélectionnant l'espace \mathcal{H}_{d^*} qui offre la meilleure garantie de risque (selon les équations (17.16) et (17.17), soit plus précisément les équations (17.13) et (17.15)). En faisant l'hypothèse que l'intervalle de confiance donné par ces équations est serré, on peut espérer ainsi obtenir une bonne approximation du risque réel associé aux hypothèses choisies, et donc pouvoir sélectionner la meilleure d'entre elles en connaissance de cause.
-
4. Plus précisément sur les classes de fonctions de perte associées. Mais pour ne pas surcharger les concepts généraux, nous n'en tenons pas compte ici. Bien sûr, dans la pratique, il faudra faire attention à ce « détail ». (Voir par exemple [Vap95, Vap98].)

Discussion

De nombreuses variantes et implémentations de l'idée générale exposée ci-dessus ont été proposées et testées (voir par exemple [BBM96], [KMNR95] pour une excellente étude comparative de plusieurs méthodes de sélection de modèles, [LZ96, Mei97, MM96, STBWA96, STBWA98, YB98]). Le principe *SRM* et l'idée essentielle de pénalisation ou de régularisation peuvent être appliqués à de nombreuses classes de modèles (*e.g.* fonctions polynomiales de degré variable, perceptrons multicouche, fonctions trigonométriques, fonctions de Fourier, etc.) ainsi qu'à des procédures d'apprentissage elles-mêmes (*e.g.* le choix des conditions initiales d'un réseau de neurones, le choix du critère d'arrêt, etc. Voir pour une bonne revue [CM98] pp.115-119). Cela met en évidence deux points importants pour la mise en pratique de l'approche *SRM*. Premièrement, le choix de la classe de modèles n'est pas spécifié par le principe *SRM* et fait partie des choix résultant de connaissances *a priori* sur le domaine ou de biais subjectifs de l'expérimentateur. Deuxièmement, si la procédure d'apprentissage elle-même peut présenter des aspects qui introduisent des facteurs de régularisation, il n'est plus possible de ne considérer que la seule régularisation introduite par la dimension de Vapnik-Chervonenkis. Cela complique alors considérablement l'estimation du bon facteur de régularisation, et explique les approches cherchant à déterminer celui-ci de manière empirique en fonction des conditions particulières d'apprentissage.

Par ailleurs, il existe des critères théoriques sur les structures de classes d'hypothèses ($\mathcal{H}_1 \subseteq \mathcal{H}_2 \subseteq \dots \subseteq \mathcal{H}_d \subseteq \dots$) dictant comment il faut régler la séquence $d_{\mathcal{H}_1} \leq d_{\mathcal{H}_2} \leq \dots \leq d_{\mathcal{H}_d} \leq \dots$ pour que la convergence vers une bonne approximation de la fonction cible f soit rapide en fonction de l'indice $d_{\mathcal{H}_d}$. Ce réglage dépend de la taille de l'échantillon d'apprentissage. L'idée principale est qu'au fur et à mesure qu'augmente la dimension $d_{\mathcal{H}_d}$ de l'espace d'hypothèse, il faut aussi augmenter la « régularité » (souvent mesurée par le degré de dérivable) des fonctions hypothèses. La conséquence est que si la fonction cible n'est pas très « régulière », il n'est pas possible de garantir une convergence rapide des espaces d'hypothèses \mathcal{H}_d vers la fonction cible. Vapnik (dans [Vap95] pp.97-100) propose alors une idée intéressante d'approximation locale, en certains voisinages d'intérêts, de la fonction cible pour garantir une convergence rapide de la capacité d'approximation. Il s'agit là d'une idée très séduisante et qui mérite des travaux complémentaires.

17.2.2 La théorie de la régularisation

Nous avons vu que l'analyse du problème de l'induction montre que le principe naïf de minimisation du risque empirique (*ERM*) est insuffisant et qu'il faut l'amender pour tenir compte de la « capacité » de l'espace d'hypothèses \mathcal{H} à décrire des fonctions quelconques. La théorie de la régularisation prescrit le même remède mais en partant d'un souci différent.

D'abord, il faut noter que l'induction d'une fonction f à partir d'un échantillon \mathcal{S} de données en nombre limité peut être vue comme un *problème inverse*. Le problème direct correspondrait à chercher l'image inconnue \mathbf{u}_i d'une valeur \mathbf{x}_i par une fonction f connue. C'est généralement un problème simple. Le problème inverse consiste à chercher une fonction f inconnue qui rende compte des couples de valeurs $\mathcal{S} = \{\mathbf{z}_1 = (\mathbf{x}_1, \mathbf{u}_1), \dots, \mathbf{z}_m = (\mathbf{x}_m, \mathbf{u}_m)\}$. Il s'agit, selon les mathématiciens, d'un *problème mal posé*. Rappelons qu'un problème bien posé (au sens de Hadamard) présente les propriétés suivantes :

1. *Existence*: pour toute fonction cible f ayant engendré les données, il existe une fonction h dans l'espace \mathcal{H} de fonctions considéré solution du problème inverse.
2. *Unicité*: la solution h est unique.
3. *Continuité*: la solution h dépend continûment de f .

L'induction est un problème mal posé dans la mesure où la solution obtenue par minimisation du risque empirique n'est en général pas unique. Par exemple, il existe une infinité de polynômes (de degré suffisamment grand) passant par un nombre fixe de points, et annulant donc le risque empirique (mesuré par exemple par un écart quadratique) (voir figure 17.7). Il peut également être mal posé dans le cas où les données sont bruitées ou engendrées par un mécanisme si complexe que la classe \mathcal{H} des hypothèses ne permet pas de trouver une fonction rendant parfaitement compte des données (non existence).

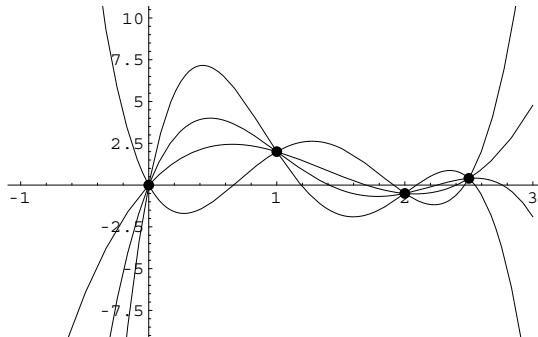


FIG. 17.7 – *Par plusieurs points passent une infinité de polynômes (ici, seuls quatre ont été tracés). Lequel doit-on choisir pour interpoler à des données inconnues ?*

La théorie de la régularisation, initiée par Tikhonov et Arsenin (1977) et développée en particulier par Poggio et Girosi [GJP95] dans le cadre de l'apprentissage, consiste à transformer le problème de l'induction en un problème bien posé (et si possible facile à résoudre effectivement) en utilisant des connaissances *a priori* pour contraindre l'espace des hypothèses.

17.2.2.1 Le principe général

La théorie de la régularisation suggère de transformer le problème mal posé de la recherche d'une hypothèse h rendant compte des données d'apprentissage \mathcal{S} en un problème de recherche d'une hypothèse h soumise à des contraintes additionnelles, c'est-à-dire :

1. minimisant le risque empirique $R_{Emp}(h)$,
2. et soumis à une contrainte $\Phi(f) \leq \mu$ où Φ est une fonctionnelle incorporant des connaissances *a priori* sur la solution recherchée et μ est un paramètre.

Sous des conditions assez larges sur Φ , il est possible de montrer que la solution au problème de minimisation ci-dessus existe, est unique et dépend continûment des données.

La question est alors de savoir *quelle forme de connaissance a priori, traduite par la fonctionnelle Φ , il faut imposer?*

Intuitivement, l'idée est encore une fois de contraindre l'espace des hypothèses en pénalisant la classe des hypothèses si complexes qu'elles peuvent s'accorder à n'importe quel échantillon de données de taille m . Deux approches sont utilisées :

1. *L'approche paramétrique* dans laquelle on cherche à contraindre le nombre de paramètres des hypothèses. Par exemple, on cherchera des réseaux connexionnistes à petit nombre de connexions.
2. *L'approche non paramétrique* qui caractérise la complexité d'une fonction hypothèse h par une mesure de sa dynamique dans le domaine fréquentiel. On parle alors de la régularité de la fonction (le terme anglais utilisé est *smoothness*). En un sens, il s'agit de préférer les fonctions les plus « lisses » parmi toutes celles qui rendent compte des données. (Par exemple, dans le cas des polynômes, on favorisera les polynômes de degré plus faible).

17.2.2.2 La méthode des multiplicateurs de Lagrange

La minimisation des problèmes sous contrainte du type:

$$\begin{cases} \text{minimiser une fonctionnelle: } F(h) \\ \text{sous la contrainte: } G(h) \leq \mu \end{cases} \quad (17.18)$$

se résout en faisant appel à la méthode des multiplicateurs de Lagrange.

On construit d'abord le problème d'optimisation sous-contraint:

$$R_{Pen}(h) = F(h) + \lambda G(h) \quad (17.19)$$

où λ est un paramètre portant le nom de multiplicateur de Lagrange. Le point selle de cette fonctionnelle fournit alors la solution du problème d'optimisation. La fonctionnelle doit être minimisée en fonction de h et maximisée en fonction de λ .

La solution canonique du problème d'optimisation ci-dessus passe par deux étapes:

1. Pour chaque valeur de $\lambda > 0$, trouver le minimum $m(\lambda)$ du problème sous-contraint (17.19)
2. Trouver la valeur $\lambda = \hat{\lambda}$ pour laquelle: $G(m(\lambda)) = \mu$.

Le minimum du problème constraint est: $m(\hat{\lambda})$.

17.2.2.3 Le réglage du multiplicateur de Lagrange

Généralement cependant, on applique la théorie de la régularisation différemment. On choisit une fonctionnelle de pénalisation $G(h)$ correspondant à des connaissances *a priori* sur les hypothèses souhaitables, et compatible avec les techniques efficaces d'optimisation. On cherche alors une hypothèse h minimisant:

$$R_{Pen}(h) = R_{Emp}(h) + \lambda G(h) \quad (17.20)$$

λ agit comme un paramètre de contrôle permettant de régler le compromis entre la fidélité aux données d'apprentissage mesurée par le premier terme de l'équation, et la régularité de la solution h mesurée par le second terme. Cela définit un nouveau principe inductif: **choisir une hypothèse h minimisant le risque pénalisé**.

Normalement, tant la fonctionnelle de pénalisation G que le paramètre λ devraient traduire des connaissances *a priori*, c'est-à-dire *externes* aux données d'apprentissage. Cependant, en général, il est difficile d'avoir suffisamment d'information *a priori* sur la bonne classe d'hypothèses, et l'on ajuste donc également en partie le terme de pénalisation, ce qui permet de corriger une mauvaise estimation *a priori* mais constitue aussi une erreur en terme d'induction. On risque en effet à nouveau un phénomène de surapprentissage. Le plus souvent, la fonctionnelle G est choisie *a priori*, tandis que le paramètre λ est ajusté en fonction des données d'apprentissage.

Une procédure classique consiste à sélectionner plusieurs classes d'hypothèses (ou plusieurs valeurs du paramètre de contrôle λ) et à réaliser l'apprentissage dans chacune des classes. On sélectionne alors la classe (ou la valeur de λ) qui minimise le risque mesuré sur l'échantillon de test. Il faut cependant être conscient que le risque mesuré sur l'échantillon de test est nécessairement optimiste, puisque cet échantillon est juge et partie, ayant servi à la fois à estimer les performances des classes de modèles et à en sélectionner une. Pour avoir une estimation non biaisée du risque réel, il faut donc avoir recours à un échantillon de données n'ayant servi ni à l'apprentissage, ni au test. Ce troisième type d'échantillon est appelé *échantillon de validation*.

17.2.2.4 Applications de la méthode de régularisation

Le principe de régularisation a été employé pour justifier, souvent *a posteriori*, des méthodes visant à contrôler l'application du principe de minimisation du risque empirique dont la tendance au surapprentissage était connue des praticiens. Nous évoquons rapidement ici certaines de ces méthodes renvoyant le lecteur à des ouvrages spécialisés pour plus de détails (par exemple [Bis95]).

- *Pénalisation des poids dans un réseau connexionniste (weight decay).* Cette méthode consiste à pénaliser les réseaux connexionnistes dont les poids des connexions sont élevés. Il faut en effet des poids importants pour qu'un réseau puisse présenter une forte dynamique apte à s'adapter à n'importe quel échantillon de données. Le terme de régularisation est de la forme :

$$\lambda \sum_{\text{connexions } j} w_j^2$$

- *Règle d'arrêt avant terme (early stopping rule).* Cette méthode est appropriée dans le cadre de l'apprentissage par optimisation itérative du risque telle qu'elle s'effectue dans les réseaux connexionnistes, et en particulier les perceptrons multicouche. Elle consiste à stopper le processus d'optimisation avant qu'il y ait convergence vers l'optimum local. La justification invoquée est que le nombre effectif de degrés de liberté du réseau s'accroîtrait au fur et à mesure de l'apprentissage. En le stoppant avant terme, on éviterait l'obtention d'un modèle trop complexe.
- *Apprentissage avec bruit.* Son principe est de bruiter les données d'apprentissage durant l'apprentissage (éventuellement en les bruitant différemment à chaque passe). Intuitivement, il s'agit de rendre plus difficile l'obtention d'un modèle suffisamment complexe pour s'accorder aux données d'apprentissage. On espère ainsi que l'apprentissage résultera dans un modèle rendant compte des régularités profondes plutôt que des détails. [Bis95] (pp.346-349) montre que l'on peut effectivement exhiber une procédure de régularisation équivalente, ce qui justifie l'approche.

17.2.3 La théorie de l'estimation bayésienne

On va supposer ici que la fonction de densité de probabilité $p(\mathbf{x}|\omega)$ dépend d'un ensemble de paramètres que nous noterons $\boldsymbol{\theta} = (\theta_1, \dots, \theta_L)^\top$. Dans le cas d'un problème de classification, nous aurons une fonction par classe afin de représenter $p(\mathbf{x}|\omega_k)$ pour chaque classe ω_k , ou plus précisément, dans l'approche paramétrique, $p(x|\boldsymbol{\theta}_k)$. Soit un échantillon de données $\mathcal{S}^k = \{x_1, \dots, x_{m_k}\}$ de taille m_k relatif à une classe ω_k . En supposant que ces données soient tirées indépendamment les unes des autres suivant la loi de distribution $p(x|\boldsymbol{\theta}_k)$, alors la densité de probabilité jointe de l'échantillon total sera :

$$p(\mathcal{S}^k | \boldsymbol{\theta}_k) = \prod_{i=1}^{m_k} p(\mathbf{x}_i | \boldsymbol{\theta}_k) \equiv \mathcal{L}(\boldsymbol{\theta}_k) \quad (17.21)$$

où $\mathcal{L}(\boldsymbol{\theta}_k)$ est appelée la *vraisemblance* du vecteur de paramètres $\boldsymbol{\theta}_k$ pour l'échantillon \mathcal{S}^k donné.

La *méthode du maximum de vraisemblance* consiste à prendre pour valeur du vecteur de paramètres inconnus $\boldsymbol{\theta}_k$ celle qui maximise la vraisemblance que les données aient été produites à partir de la distribution $p(x|\boldsymbol{\theta}_k)$.

$$\hat{\boldsymbol{\theta}}_k = \underset{\boldsymbol{\theta}_k \in \Theta}{\text{ArgMax}} \mathcal{L}(\boldsymbol{\theta}_k) \quad (17.22)$$

Cette méthode consiste donc à identifier l'hypothèse (ici une classe) apparemment la meilleure, c'est-à-dire la plus vraisemblable après observation des données, puis à l'utiliser pour faire des prédictions sur des formes d'entrées non vues.

On peut cependant chercher à résoudre directement le problème de la prédiction de la valeur y_i correspondant à l'observation x_i , sans passer par le calcul préalable de l'hypothèse la plus vraisemblable. Pour cela, il existe une approche conceptuellement très intéressante et idéalement optimale, même si elle est difficile à mettre en pratique et nécessite de fait de nombreuses approximations.

L'idée essentielle est la suivante. Au lieu de chercher la valeur optimale du vecteur de paramètres θ , en maximisant la fonction de vraisemblance obtenue à partir des données, comme dans la méthode du maximum de vraisemblance, *on décrit les paramètres comme des distributions de probabilités*. Celles-ci sont initialement fixées sous forme d'une distribution *a priori*, puis transformées en distribution *a posteriori*, par l'utilisation du théorème de Bayes mis en oeuvre grâce aux données d'apprentissage. Au lieu donc de chercher une valeur spécifique de θ , nous cherchons ici à trouver la distribution de valeurs s'adaptant le mieux aux données. La prédiction pour l'événement x se fait alors par pondération des valeurs prédites pour chaque valeur de θ pondérée par la probabilité *a posteriori* de cette valeur. Nous renvoyons au chapitre 14 pour les détails de la mise en oeuvre.

Cette idée de vote, pondéré ou non, d'hypothèses se retrouve dans d'autres contextes : par exemple le *boosting*, la généralisation empilée, les méthodes d'ensemble, etc.

Ainsi l'approche de la prédiction bayésienne calcule une moyenne pondérée sur toutes les valeurs de θ au lieu de choisir une valeur spécifique. Cependant si la densité *a posteriori* $p(\theta|\mathcal{S})$ présente un pic étroit centré sur une valeur $\hat{\theta}$, alors $p(\theta|\mathcal{S}) \approx p(h|\hat{\theta})$, et nous retrouvons le résultat donné par la méthode du maximum de vraisemblance. Cela arrive généralement pour les échantillons d'apprentissage de grande taille.

Bien que cela ne soit pas le sujet de ce chapitre, il est utile de noter que le principe du maximum de vraisemblance et l'apprentissage bayésien ne se prêtent pas aux mêmes méthodes de calcul. Le premier se traite comme un problème d'optimisation : il faut chercher le minimum d'une fonction d'erreur. En revanche, dans le second, l'essentiel du calcul implique une intégration sur des espaces de grandes dimensions. Dans ce dernier cas, les méthodes classiques d'intégration ne conviennent pas, et il faut se tourner vers des méthodes approchées, par exemple les méthodes de Monte-Carlo (voir le chapitre 3).

17.3 L'induction par compression d'information

Un principe inductif intuitif consiste à préférer parmi les hypothèses capables d'expliquer les données d'apprentissage celle qui est la plus simple, c'est-à-dire dont l'expression est la plus économique. Cela correspond au principe du rasoir d'Occam (voir la page 118). Il repose sur deux idées fondamentales. La première selon laquelle apprendre quelque chose à partir de données signifie identifier des régularités sous-jacentes. La seconde que l'identification de régularités permet de comprimer l'expression des données. On en conclut donc que plus il est possible de comprimer l'expression de données (sans perdre d'information), mieux on connaît ces données. Une troisième idée relève de l'induction, à savoir que lorsque l'on sait quelque chose à propos d'un échantillon de données, il est possible d'utiliser cette connaissance pour prédire de futures données. Ainsi, la compression et la prédiction, c'est-à-dire l'induction, semblent intrinsèquement liées.

17.3.1 Un exemple

Supposons que nous étudions un phénomène caractérisé par des séquences d'observations : 0 avec une étiquette '+' ou '-' fournie par un oracle. Nous décidons d'en rendre compte à l'aide d'un automate déterministe à états finis (*DFA* : *Deterministic Finite state Automaton*). La simplicité (ou plutôt sa complexité) d'un automate sera mesurée par son nombre d'états.

Les séquences suivantes sont positives, ce qui correspond au fait qu'elles sont reconnues par l'automate recherché : 0, 000, 00010, 00000000 ; les séquences suivantes sont négatives, donc rejetées par l'automate : ϵ , 00, 0000, 000000.

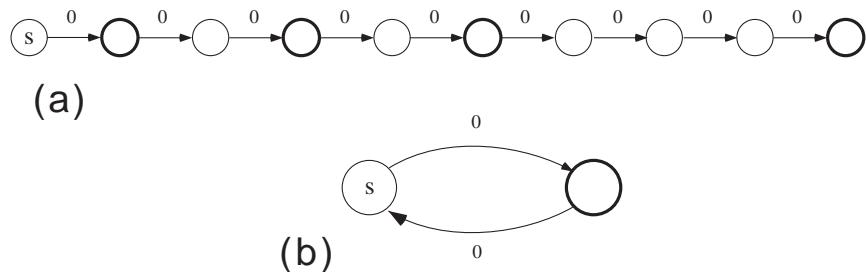


FIG. 17.8 – En (a) l'automate trivialement cohérent avec les données. En (b) l'automate le plus simple cohérent avec les données.

Il existe une infinité d'automates cohérents avec ces séquences. La figure 17.8(a) montre l'automate trivial qui code simplement ces séquences. La figure 17.8(b) montre l'automate le plus simple permettant de rendre compte de ces données. Lequel des deux devont nous préférer pour prédire l'étiquette de nouvelles séquences d'observations ? Notre intuition nous souffle que le second est meilleur. Avons-nous raison ?

17.3.2 La théorie de l'induction selon Solomonoff

Dans un papier visionnaire, Solomonoff en 1964 [Sol64] a proposé une formalisation du problème de l'induction. Selon lui, tout problème d'inférence inductive peut être considéré comme un problème d'extrapolation d'une séquence de symboles binaires. Soit l'espace \mathcal{S} des séquences infinies de symboles binaires, et une distribution *a priori* μ sur \mathcal{S} , avec $\mu(\mathbf{x})$ dénotant la probabilité d'une séquence commençant avec \mathbf{x} . Alors, étant donnée une séquence observée \mathbf{x} , le problème inductif est de prédire le prochain symbole dans la séquence. Cela peut se faire soit par prédiction directe du prochain symbole, soit par identification d'une règle sous-jacente à la séquence permettant de prédire le prochain symbole. On peut exprimer la probabilité que la séquence \mathbf{x} se poursuive par le symbole a sachant que la séquence initiale est \mathbf{x} par :

$$\mu(a|\mathbf{x}) = \frac{\mu(\mathbf{x}a)}{\mu(\mathbf{x})} \quad (17.23)$$

La tâche centrale de l'inférence inductive est alors de trouver une approximation de μ permettant d'estimer la probabilité conditionnelle qu'un segment \mathbf{x} soit suivi d'un segment \mathbf{y} . Ceci est dans le cas général impossible. Il faut donc trouver des moyens d'approcher μ de manière raisonnable.

17.3.3 La complexité de Kolmogorov

La complexité algorithmique, souvent appelée complexité de Kolmogorov du nom de l'un de ses inventeurs, cherche à mesurer la complexité intrinsèque d'une chaîne de bits.

Définition 17.2 (Complexité algorithmique)

La complexité algorithmique d'une chaîne de bits \mathbf{x} est définie comme la longueur (mesurée en bits) du plus court programme qui, sans données supplémentaires, permet à une machine de Turing universelle \mathcal{U} d'écrire la chaîne \mathbf{x} et de s'arrêter.

Formellement, cela s'écrit :

$$K(\mathbf{x}) = \underset{l(p)}{\text{Min}}[\mathcal{U}(p) = \mathbf{x}] \quad (17.24)$$

où $l(p)$ est la longueur, mesurée en bits, du programme p .

La complexité algorithmique est une mesure de l'incompressibilité de \mathbf{x} . Considérons par exemple une chaîne \mathbf{x} constituée uniquement de n 1. Cette chaîne est intuitivement très simple. Et de fait, il est facile d'écrire un programme pour la produire. Ce programme est essentiellement une boucle qui sera exécutée n fois. Le programme est donc de longueur proportionnelle à $\log_2 n$ soit en $O(\log_2 n)$. Nous noterons cette complexité de Kolmogorov par $K(\mathbf{x}) = \log_2 n$. Un autre exemple est celui de l'expression du nombre transcendantal π , dont la séquence binaire : 11.0010010001111101101010001... apparemment aléatoire est en fait simple : la taille du plus petit programme capable de produire cette séquence est petite et constante, indépendante du nombre de bits produits. On a donc $K(\pi) = 1$. En revanche, une séquence réellement aléatoire ne peut pas être produite par un programme plus court que la séquence elle-même. Dans ce cas, on a donc : $K(\mathbf{x}) = |\mathbf{x}|$.

Sans faire justice de toutes les implications et subtilités de la théorie de la complexité algorithmique (nous renvoyons pour cela le lecteur à la « somme » de Li et Vitanyi [LV97]), il suffit pour nous de savoir qu'elle est liée profondément à une mesure de probabilité universelle. En effet, on peut associer à chaque programme p , c'est-à-dire chaîne de bits, sa probabilité de production par un tirage aléatoire de bits avec probabilité $1/2$. Cette probabilité est : $Pr(p) = 2^{-l(p)}$, où $l(p)$ est la longueur de la chaîne de bits correspondant à p . Cela signifie qu'un programme court est plus probable qu'un programme long. Si un programme court produit une séquence longue, celle-ci ne peut être aléatoire puisqu'elle a une description simple. On est ainsi amené à définir la probabilité universelle d'une chaîne \mathbf{x} par :

$$P_{\mathcal{U}}(\mathbf{x}) = \sum_{p: \mathcal{U}(p)=\mathbf{x}} 2^{-l(p)} \quad (17.25)$$

C'est la probabilité qu'un programme p tiré aléatoirement suivant une distribution de probabilité $1/2$ produise la suite \mathbf{x} par la machine de Turing universelle \mathcal{U} . Cette probabilité, sans être exactement indépendante de la machine employée, en dépend relativement peu. Par ailleurs, il est clair que cette probabilité est dominée par la probabilité $2^{-K(\mathbf{x})}$ du plus court programme pouvant produire la séquence \mathbf{x} . Malheureusement, la taille $K(\mathbf{x})$ de ce plus court programme est non calculable effectivement. En effet, le seul moyen de trouver ce plus court programme serait d'essayer tous les programmes possibles, or l'exécution de certains d'entre eux risque de ne jamais se terminer. Il semble ainsi que nous soyons ramenés au problème précédent : nous ne pouvions définir une mesure de probabilité universelle μ , et nous ne pouvons pas plus définir la complexité algorithmique d'une chaîne de bits qui aurait pu nous permettre de calculer μ . Cependant, ce lien très profond entre mesure de probabilité et complexité algorithmique a fourni le terreau sur lequel ont été développés plusieurs procédés inductifs de nature plus heuristique.

17.3.4 Le principe de longueur de description minimale (*MDLP*)

Le principe de longueur minimale de description, (*Minimum Description Length principle* ou *MDLP*), peut s'expliquer par une analogie avec la théorie de l'information et la transmission de message entre un émetteur et un récepteur.

Supposons qu'un agent, appelé émetteur, veuille transmettre des données à un autre agent, appelé récepteur, de la manière la plus économique, c'est-à-dire en limitant autant que possible le nombre de bits transmis sur le canal qui les relie (une ligne téléphonique par exemple). Intuitivement, une manière de faire consisterait à d'abord transmettre une description générale des données (*e.g.* « je viens de voir passer deux oiseaux style corbeau ») puis à transmettre ce qui dans les données ne correspond pas au modèle, c'est-à-dire les exceptions (*e.g.* « sauf que l'un avait le bec jaune et l'autre le bout des ailes rouges »). Il existe bien sûr un compromis entre la complexité du modèle transmis et ce qu'il faut indiquer comme exception. Si le modèle transmis est très général (*em e.g.* « j'ai vu des objets volants »), il faudra fournir beaucoup d'informations pour décrire exactement les données à partir de ce modèle. Inversement, si le modèle est très précis (*e.g.* « j'ai vu un oiseau noir, de la taille d'un corbeau environ, de plumage noir avec le bec jaune, et un autre oiseau ... »), il sera coûteux à transmettre, ne factorisant pas les généralités présentes dans les données. Le meilleur compromis consiste à trouver un modèle tel que la somme de sa description, et celle des irrégularités par rapport à ce modèle, soit la plus économique possible. C'est l'essence du principe de longueur de description minimale.

Définition 17.3 (Principe de longueur minimale de description (*MDLP*))

La meilleure théorie, ou hypothèse, ou le meilleur modèle, rendant compte d'un échantillon d'apprentissage minimise la somme de :

1. *la longueur, mesurée en bits, de la description de la théorie; et de*
2. *la longueur, mesurée en bits, de la description des données lorsqu'elles sont décrites à l'aide de la théorie.*

Formellement, cela signifie que l'hypothèse optimale h^ vérifie :*

$$h^* = \underset{h \in \mathcal{H}}{\operatorname{ArgMin}} L(h) + L(\mathbf{x}|h) \quad (17.26)$$

où $L(h)$ mesure la longueur de description de h , et $L(\mathbf{x}|h)$ mesure la longueur de description des données \mathbf{x} en utilisant l'hypothèse h pour les coder.

Avec ce principe, on retrouve l'idée essentielle des principes inductifs contrôlant la complexité des hypothèses, à savoir qu'il faut faire place à un compromis entre la complexité de l'espace d'hypothèses mis en œuvre pour rendre compte des données et la fidélité aux données elles-mêmes. Un modèle des données trop précis peut ne pas avoir de bonnes performances en généralisation, de même qu'un modèle trop général.

Il est facile de voir que le principe de longueur minimale de description est lié à la règle de Bayes. En effet, d'après cette règle :

$$\Pr(h|\mathbf{x}) = \frac{\Pr(\mathbf{x}|h) \Pr(h)}{\Pr(\mathbf{x})}$$

Soit, en prenant l'opposé du logarithme de chaque côté de l'équation :

$$-\log \Pr(h|\mathbf{x}) = -\log \Pr(\mathbf{x}|h) - \log \Pr(h) + \log \Pr(\mathbf{x})$$

En tenant compte du fait que le facteur $Pr(\mathbf{x})$ est indépendant de l'hypothèse mise en œuvre pour rendre compte des données, maximiser $Pr(h|\mathbf{x})$, comme le préconise le principe du maximum de vraisemblance, revient à minimiser le terme de droite de l'équation précédente, soit :

$$-\log Pr(\mathbf{x}|h) - \log Pr(h)$$

Idéalement, la mesure de probabilité à utiliser serait la mesure universelle μ et donc celle de la complexité algorithmique K . C'est-à-dire qu'il faudrait prendre $Pr(\mathbf{y})$ comme étant égal à $2^{-K(\mathbf{y})}$ pour une séquence arbitraire \mathbf{y} . On choisirait alors l'hypothèse h minimisant :

$$K(\mathbf{x}|h) + K(h)$$

À défaut de pouvoir utiliser la mesure de probabilité μ ou la mesure de complexité algorithmique K , le principe de longueur minimale de description préconise de définir un « codebook » raisonnable permettant la description de l'univers considéré, puis de coder les hypothèses et les données à l'aide de ce codebook. La mesure de complexité, ou de longueur de description, se fait alors en référence au code ainsi défini, chaque élément de ce code étant associé à un coût fixé par l'utilisateur. On retrouve alors l'équation (17.26) relative au *MDLP*.

Exemple 16 (Régression par des polynômes)

On suppose que l'on a un échantillon de données $\mathcal{S} = \{(x_1, u_1), (x_2, u_2), \dots, (x_m, u_m)\}$, dans lequel les formes x_i et les étiquettes u_i sont des nombres réels. On cherche à pouvoir prédire la valeur u pour une forme x donnée. Pour cela, on fait l'hypothèse que l'on peut rendre compte des données avec un polynôme. En général, plus le degré du polynôme est élevé, plus l'adéquation aux données est étroite (au sens par exemple des moindres carrés). À la limite, pour tout ensemble de m points (x_i, u_i) , il est possible de trouver un polynôme de degré $m-1$ passant exactement par les m points. Mais ce polynôme n'aura en général aucun pouvoir prédictif (le lecteur peut s'en convaincre aisément en essayant de prédire les cours de la bourse de cette manière). Supposons que l'on cherche une hypothèse sous la forme d'un polynôme p_k de degré k . Pour décrire un tel polynôme, il faut $k+1$ coefficients que nous supposerons décrits avec une précision de d bits. Une hypothèse, c'est-à-dire un polynôme, sera alors décrite par : $k d + \mathcal{O}(\log k d)$ bits. (Le deuxième terme de la somme provient de considérations techniques sur le fait que le programme à fournir à la machine de Turing doit être autodélimitant.)

Il faut maintenant examiner le coût de description des données (x_i, u_i) à l'aide d'un polynôme. En général, on fait comme si le polynôme correspondait au vrai modèle sous-jacent aux données et que celles-ci étaient distribuées suivant une loi gaussienne autour de la valeur prédite par le polynôme : $u_i = p_k(x_i) + \varepsilon$ avec ε une variable centrée en $p_k(x)$ et de variance constante. Dans ce cas la probabilité d'observer la valeur u_i au lieu de $p_k(x_i)$ est de l'ordre de $e^{-(p_k(x_i) - u_i)^2}$. À l'aide la mesure de probabilité universelle, cette grandeur est codée à l'aide de $s(p_k(x_i) - u_i)^2$ bits, où s est une constante de normalisation.

Comme l'erreur commise par l'hypothèse p_k au sens des moindres carrés est :

$$\text{erreur}(p_k) = \sum_{i=1, m} (p_k(x_i) - u_i)^2$$

on trouve, en négligeant le terme $\mathcal{O}(\log k d)$ que le coût d'expression des données \mathcal{S} à l'aide du polynôme p_k est de :

$$k d + s \cdot \text{erreur}(p_k)$$

La meilleure hypothèse est, selon le principe de minimisation de longueur de description (*MDLP*), le polynôme (p_k qui minimise cette expression). \square

Le principe *MDLP* a été appliqué dans de nombreux autres contextes. Nous citerons par exemple le choix d'arbres de décision (voir chapitre 11) pour lequel Quinlan et Rivest ([QR89]) ont proposé une mesure de coût des arbres prenant en compte les nœuds de l'arbre et les branchements possibles, ainsi que les exceptions. Dans un contexte différent de celui de l'induction, [Cor96] propose d'utiliser une version du *MDLP* pour rendre compte du raisonnement par analogie, considéré comme une forme de transmission économique d'information. Si le *MDLP* a permis d'obtenir des résultats dans plusieurs applications, il n'en reste pas moins une technique largement empirique, dans laquelle on a remplacé la nécessité de fournir des probabilités *a priori*, comme dans l'approche bayésienne, par celle de concevoir un codebook avec ses coûts associés. Pour le moment on ne connaît pas de technique fondée rigoureusement pour résoudre ce problème. Par ailleurs, la recherche d'une description la plus courte est connue comme étant un problème NP-complet dans de nombreux formalismes. Il est donc nécessaire d'avoir recours à des techniques heuristiques de recherche d'hypothèses.

17.3.5 Analyse : compression et pouvoir inductif

Si intuitivement il semble satisfaisant de penser qu'un modèle « simple » des données est plus susceptible qu'un modèle complexe de décrire les régularités sous-jacentes, et donc de permettre des prédictions, cela ne suffit pas à garantir le lien entre compression des informations et pouvoir inductif. Sans entrer dans les détails de ce lien ici, nous mentionnerons deux études se rapportant à cette question.

17.3.5.1 Un théorème de Vapnik

Dans son livre de 1995 [Vap95] en pp.102-105, Vapnik donne une preuve justifiant le principe de longueur minimale de description pour la tâche de classification. Il montre que le coefficient de compression obtenue, c'est-à-dire le rapport $r(h)$ entre la taille de description comprimée des données et la taille de leur description brute, est lié à la probabilité d'erreur de classification sur des données futures. La preuve, sans être difficile, dépasse le cadre du présent ouvrage, et nous n'en donnons que le résultat. Elle repose sur un argument de convergence uniforme appliquée au codebook utilisable pour décrire l'espace des fonctions de classification.

Théorème 17.5 (*MDLP* et probabilité d'erreur en classification (Vapnik,95))

Si, en utilisant un codebook structuré, on trouve une hypothèse h permettant de comprimer l'expression de la chaîne de bits u_1, u_2, \dots, u_m des étiquettes des formes d'apprentissage $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m$ d'un facteur $R(h)$, alors, avec probabilité au moins $1 - \eta$, la probabilité d'erreur de classification par l'hypothèse h est bornée par :

$$R_{\text{Réel}}(h) < 2(r(h) \ln 2 - \frac{\ln \eta}{m})$$

Ce théorème est intéressant dans la mesure où, par contraste avec les théorèmes de pertinence de l'*ERM*, il ne fait pas intervenir directement de propriétés statistiques des données, ni de risque empirique (nombre d'erreurs de classification en apprentissage). Malheureusement, ce théorème ne dit pas comment construire un bon codebook. Nous en verrons une raison en analysant le *No-Free-Lunch theorem* (17.4.1).

17.3.5.2 Les algorithmes d'Occam en apprentissage PAC

Le lien entre compression d'information et généralisation a été également étudié dans le cadre de l'apprentissage de fonctions indicatrices, c'est-à-dire de concepts. Dans ce cadre, en supposant que les exemples \mathbf{x} soient définis sur $\{0, 1\}^d$ ou sur \mathbb{R}^d , que l'échantillon d'apprentissage \mathcal{S} comporte m exemples étiquetés suivant une fonction cible $f : ((\mathbf{x}_1, f(\mathbf{x}_1)), (\mathbf{x}_2, f(\mathbf{x}_2)), \dots, (\mathbf{x}_m, f(\mathbf{x}_m)))$, alors un algorithme d'Occam est un algorithme qui, prenant \mathcal{S} en entrée, produit une hypothèse $h \in \mathcal{H}$ cohérente avec \mathcal{S} et qui est succincte au sens où $\text{taille}(h)$ est une fonction croissant suffisamment lentement en fonction de d , $\text{taille}(f)$ et m . Plus précisément :

Définition 17.4 (Algorithme d'Occam)

Soient deux constantes $\alpha \geq 0$ et $0 \leq \beta < 1$. On dit qu'un algorithme d'apprentissage est un algorithme d'Occam si, à partir d'un échantillon d'apprentissage étiqueté par un concept cible f , il produit une hypothèse h vérifiant :

1. h est cohérente avec \mathcal{S}
2. $\text{taille}(h) \leq (d \cdot \text{taille}(f))^\alpha m^\beta$

Il est clair que si $m \gg d$, alors les m bits correspondants aux étiquettes de $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m$ sont effectivement comprimés en une chaîne d'au plus m^β bits. Sinon, il faut bien exprimer que l'hypothèse la plus courte peut dépendre au moins linéairement de $\text{taille}(f)$.

Il existe alors un théorème prouvant qu'un tel algorithme, si on lui fournit un échantillon de taille

$$m \geq a \left(\frac{1}{\epsilon} \log \frac{1}{\delta} + \left(\frac{(d \cdot \text{taille}(f))^\alpha}{\epsilon} \right)^{\frac{1}{1-\beta}} \right)$$

où a est une constante > 0 , produit une hypothèse h de probabilité d'erreur de classification en généralisation $\leq \epsilon$ avec une probabilité $1 - \delta \leq 1$.

Un lien est donc établi là aussi entre compression et induction, même si la portée de ce théorème est limitée par le fait qu'il ne s'applique que pour des espaces d'hypothèses \mathcal{H} de cardinal fini. (Pour plus de détails, se reporter par exemple à [KV94]).

17.3.5.3 Pour conclure

Doit-on conclure des résultats qui précèdent qu'il faut toujours mieux choisir une hypothèse succincte pour rendre compte de données ? Nous allons voir dans la section suivante que la réponse est non. Plusieurs études se voulant provocatrices ont d'ailleurs montré que le choix d'une hypothèse succincte pouvait se révéler moins bon que celui d'une hypothèse aussi bonne sur les données d'apprentissage, mais plus complexe. De fait, la préférence pour la simplicité des hypothèses s'assimile à un biais *a priori*, qui peut, ou non, être approprié. Pourquoi alors est-ce un biais naturel chez les êtres humains et que l'on trouve souvent satisfaisant ?

La réponse à cette question comporte au moins deux volets. Le premier est qu'à côté du pouvoir prédictif d'une hypothèse ou d'une théorie, nous recherchons souvent son caractère explicatif et donc compréhensible. Une hypothèse s'accordant parfaitement aux données, mais compliquée, est souvent moins satisfaisante qu'une hypothèse moins parfaite mais intelligible. À partir du moment où l'on parle d'intelligibilité, il faudrait aussi faire intervenir le reste des connaissances préalables dans lesquelles s'inscrit la nouvelle connaissance apprise. Les théorèmes de compression ne disent évidemment rien sur cet aspect des choses.

Le deuxième volet nous ramène au sens profond des théorèmes de pertinence du principe ERM. Pourquoi en effet une hypothèse simple serait meilleure en prédiction qu'une hypothèse

plus complexe s'accordant aussi bien aux données ? Rien dans les théorèmes de Vapnik ne permet de l'expliquer. Rien, sauf ceci. Que la classe des hypothèses simples à exprimer dans un langage, dans tous langages, est forcément restreinte, quel que soit le langage utilisé. Si donc l'on trouve une hypothèse « simple » qui s'accorde bien aux données d'apprentissage, c'est que, dans un espace \mathcal{H} limité, on a trouvé une bonne hypothèse au sens du risque empirique. Les théorèmes de Vapnik, qui prennent en compte la richesse de l'espace des hypothèses, affirment alors que la probabilité est grande que cette hypothèse se comporte bien à l'avenir. La simplicité d'une hypothèse est relative au langage utilisé pour l'exprimer, mais ce qui compte vraiment c'est la richesse de la classe des hypothèses à laquelle elle appartient. Si par chance on trouve une bonne hypothèse dans une classe restreinte, alors l'espérance est grande qu'elle soit bonne en général. C'est ce que Judea Pearl avait déjà remarqué dans un article de 1978 [Pea78] injustement oublié et qui préemptait bien des travaux ultérieurs sur ce sujet.

17.4 L'induction en débat

Ce chapitre a discuté plusieurs principes inductifs et leurs variations nées de l'étude des conditions de leur validité. Ainsi, ont été passés en revue le principe *ERM* favorisant les hypothèses qui s'accordent le mieux aux données d'apprentissage, le principe bayésien stipulant (dans sa version maximum de vraisemblance) de choisir l'hypothèse dont il est le plus probable qu'elle soit à l'origine des données, le principe de compression d'information prescrivant de choisir le modèle du monde conduisant à sa description la plus compacte. Le chapitre 14 a également étudié un principe inductif classique qui est celui de prédire la réponse en un point en prenant la réponse majoritaire en des points proches. Nous avons également vu que l'étude théorique de ces principes avait conduit à des principes inductifs plus sophistiqués dans lesquels la richesse de l'espace d'hypothèses est prise en compte. Les recherches récentes, portant en particulier sur les séparateurs à vastes marges (SVM) raffinent ces principes en prescrivant de prendre en compte aussi la distribution des exemples. Une question naturelle est alors de se demander lequel de ces principes inductifs est le meilleur ; lequel nous devrions choisir. Bien entendu, ces principes et leurs réalisations sous forme d'algorithmes, peuvent présenter des caractéristiques computationnelles différentes, par exemple des coûts computationnels polynomiaux pour certains, exponentiels pour d'autres, ou bien peuvent conduire à des résultats plus ou moins intelligibles, etc. Tous ces facteurs sont importants en pratique, mais si, provisoirement, on ne s'intéresse qu'à la performance en généralisation, l'espérance de risque, quel principe inductif est le meilleur ? Devons-nous par exemple favoriser les méthodes de minimisation du risque structurel (*SRM*) de Vapnik, ou bien devons-nous chercher les hypothèses les plus économiques ? En bref, existe-t-il un principe qui soit meilleur que les autres en général, indépendamment du problème étudié ?

17.4.1 Le *no-free-lunch theorem* : toutes les méthodes se valent !?

Le chapitre 1 a déjà apporté des éléments de réponse à cette question en insistant sur la nécessité d'un biais d'apprentissage pour permettre l'induction, c'est-à-dire d'hypothèses *a priori* sur le monde. Un théorème formalise et généralise cette idée : le *no-free-lunch theorem* dû à Wolpert (1992) [Wol92]. Selon ce théorème, tous les principes inductifs, et tous les algorithmes d'apprentissage se valent. En l'absence de toute information sur le problème d'apprentissage autre que l'échantillon de données, aucune méthode n'est meilleure qu'une autre, y compris celle qui consiste à tirer une hypothèse au hasard. Exprimé d'une autre manière, ce théorème affirme qu'il n'y a *a priori* aucune corrélation entre l'échantillon de données \mathcal{S} observé et les événements non encore observés. De ce fait, toute hypothèse sélectionnée sur la base de \mathcal{S} n'a aucune raison

d'être performante à l'avenir en dehors de \mathcal{S} . De manière plus abrupte, en dehors d'information supplémentaire sur le problème d'apprentissage, c'est-à-dire sur l'espace des fonctions cible, il n'est pas possible de faire autre chose que de l'apprentissage par cœur ! Aucune induction n'est possible, ou, du moins, légitime.

Avant d'examiner une expression plus formelle de ce théorème, essayons d'en saisir l'intuition. Soit l'espace \mathcal{F} des fonctions cible. Soit \mathcal{X} l'espace des entrées, et soit \mathcal{U} l'espace des sorties. On suppose qu'un échantillon de formes $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m$ soit tiré aléatoirement suivant une distribution $d_{\mathcal{X}}$ inconnue sur \mathcal{X} . Chacune de ces formes est étiquetée pour former l'échantillon $\mathcal{S} = \{(\mathbf{x}_1, \mathbf{u}_1), (\mathbf{x}_2, \mathbf{u}_2), \dots, (\mathbf{x}_m, \mathbf{u}_m)\}$. On suppose ici que cet échantillon n'est pas bruité. Les étiquettes \mathbf{u}_i ont donc été calculées grâce à l'application d'une fonction $f \in \mathcal{F}$. Le problème de l'induction est d'estimer laquelle sur la base de l'échantillon \mathcal{S} .

En l'absence d'informations supplémentaires sur \mathcal{F} , toutes les fonctions $f \in \mathcal{F}$ sont également possibles. Une fois fixé l'échantillon d'apprentissage, un certain nombre de ces fonctions sont éliminées car ne s'accordant pas aux données, mais toutes les autres fonctions restent candidates, et aucune prédition n'est possible. C'est ce que nous avons vu dans le cas de fonctions binaires dans le chapitre 1. C'est également ce que montre la figure 17.7.

Si donc l'unique information dont nous disposons pour une tâche inductive est un échantillon d'apprentissage, alors seul un apprentissage par cœur de cet échantillon est possible, et aucune induction ne peut être faite avec quelque garantie que ce soit. En d'autres termes, et exprimé de manière peut-être plus brutale, il n'existe aucune corrélation *a priori* entre un échantillon d'apprentissage et les événements non vus. Plus formellement, notons $p(h|\mathcal{S})$ la distribution des hypothèses dans \mathcal{H} après la prise en compte de l'échantillon \mathcal{S} , c'est-à-dire après apprentissage. Si l'algorithme d'apprentissage est déterministe, fournissant une seule hypothèse, et toujours la même, pour un échantillon \mathcal{S} donné, alors la distribution prend la forme d'un Dirac centré sur l'hypothèse choisie h . Si au contraire il s'agit d'un algorithme non déterministe, $p(h|\mathcal{S})$ peut avoir une certaine extension. De la même manière, nous notons $p(f|\mathcal{S})$ la distribution de probabilité des fonctions de la Nature f étant donné l'échantillon d'apprentissage. L'expression de l'espérance de l'*« écart »* entre le résultat de l'apprentissage et la Nature est alors :

$$E[R_{R\acute{e}el}|\mathcal{S}] = \int_{h,f} \int_{\mathbf{x} \notin \mathcal{S}} p(\mathbf{x}) [1 - \delta(f(\mathbf{x}), h(\mathbf{x}))] p(h|\mathcal{S}) p(f|\mathcal{S}) \quad (17.27)$$

où le symbole de Kronecker δ dénote la fonction nulle partout sauf là où ses arguments sont égaux, où elle vaut 1. Nous noterons ici que la somme ne fait intervenir que les formes \mathbf{x} non vues en apprentissage, ce qui est différent de l'espérance de risque i.i.d. dans laquelle le tirage aléatoire des formes peut permettre le tirage de la même forme en apprentissage et en reconnaissance. Les deux expressions sont équivalentes dans le cas où l'échantillon \mathcal{S} est de mesure nulle sur l'espace des entrées possibles \mathcal{X} . L'équation 17.27 exprime que l'espérance de risque réel étant donné un échantillon d'apprentissage \mathcal{S} est liée à la somme de toutes les entrées possibles \mathbf{x} pondérées par leur probabilité $p(\mathbf{x})$, et à un *« alignement »* entre l'algorithme d'apprentissage caractérisé par $p(h|\mathcal{S})$ et la vraie probabilité *a posteriori* de la Nature $p(f|\mathcal{S})$. De ce fait, en l'absence d'information *a priori* sur la distribution $p(f|\mathcal{S})$, il est impossible de dire quoi que ce soit sur la performance en généralisation de l'algorithme d'apprentissage.

Si l'affirmation précédente n'a pas suffi à plonger le lecteur dans la consternation, le corollaire⁵ suivant devraitachever de le faire. Nous noterons :

$$E_k[R_{R\acute{e}el}|f, m] = \int_{\mathbf{x} \notin \mathcal{S}} p(\mathbf{x}) [1 - \delta(f(\mathbf{x}), h(\mathbf{x}))] p_k(h(\mathbf{x})|\mathcal{S})$$

5. Du latin *corollarium* « petite couronne donnée comme gratification ».

l'espérance de risque associée à l'algorithme d'apprentissage \mathcal{A}_k étant donné l'échantillon d'apprentissage \mathcal{S} , et la vraie fonction de la Nature f .

Théorème 17.6 (*No-free-lunch theorem (Wolpert, 1992)*)

Pour tout couple d'algorithmes d'apprentissage \mathcal{A}_1 et \mathcal{A}_2 , caractérisés par leur distribution de probabilité a posteriori $p_1(h|\mathcal{S})$ et $p_2(h|\mathcal{S})$, et pour toute distribution $d_{\mathcal{X}}$ des formes d'entrées \mathbf{x} et tout nombre m d'exemples d'apprentissage, les propositions suivantes sont vraies :

1. En moyenne uniforme sur toutes les fonctions cible f dans \mathcal{F} : $E_1[R_{R\acute{e}el}|f, m] - E_2[R_{R\acute{e}el}|f, m] = 0$.
2. Pour tout échantillon d'apprentissage \mathcal{S} donné, en moyenne uniforme sur toutes les fonctions cible f dans \mathcal{F} : $E_1[R_{R\acute{e}el}|f, \mathcal{S}] - E_2[R_{R\acute{e}el}|f, \mathcal{S}] = 0$.
3. En moyenne uniforme sur toutes les distributions possibles $p(f)$: $E_1[R_{R\acute{e}el}|m] - E_2[R_{R\acute{e}el}|m] = 0$.
4. Pour tout échantillon d'apprentissage \mathcal{S} donné, en moyenne uniforme sur toutes les distributions possibles $p(f)$: $E_1[R_{R\acute{e}el}|\mathcal{S}] - E_2[R_{R\acute{e}el}|\mathcal{S}] = 0$.

Pour une preuve de ce théorème, nous renvoyons le lecteur à [Wol92]. De manière qualitative, le premier point de ce théorème exprime que quel que soit notre choix d'un « bon » algorithme d'apprentissage et d'un « mauvais » algorithme (par exemple un algorithme prédisant au hasard, ou bien une fonction constante sur \mathcal{X}), si toutes les fonctions cible f sont également probables, alors le « bon » algorithme aura la même performance en moyenne que le « mauvais ». Cela signifie aussi qu'il existe au moins une fonction cible pour laquelle la prédiction au hasard est meilleure que n'importe quelle autre stratégie de prédiction.

Le deuxième point du théorème affirme la même absence de supériorité d'un algorithme d'apprentissage sur tout autre algorithme, même quand l'échantillon d'apprentissage est connu. En d'autres termes, celui-ci n'apporte pas plus d'information à un algorithme plutôt qu'à un autre, fût-il à nouveau l'algorithme de prédiction au hasard. Les points trois et quatre ne font que renforcer ces résultats en affirmant l'égalité de tous les algorithmes, si l'on prend en compte des distributions non uniformes de fonctions cible, mais que l'on moyenne sur toutes ces distributions. Bien sûr, pour une distribution donnée, un algorithme va être meilleur que les autres, à savoir celui qui a la même distribution que $p(f|\mathcal{S})$. Mais comment le deviner *a priori* ?

Avant de discuter des leçons à tirer du *no-free-lunch theorem*, il est utile d'en illustrer la force à nouveau sur un exemple. Nous avons là en effet une sorte de loi de conservation (comme le dit Cullen Schaffer [Sch94]). De même que pour chaque classe de problèmes pour laquelle un algorithme d'apprentissage est meilleur qu'un algorithme de prédiction au hasard il existe une classe de problèmes pour laquelle cet algorithme est moins bon (voir figure 17.9). De même, pour chaque algorithme d'apprentissage, il existe des problèmes pour lesquels la courbe de performance en généralisation est ascendante et des problèmes pour lesquels cette courbe est descendante, c'est-à-dire pour lesquels plus l'algorithme apprend et plus il est mauvais en généralisation !

Considérons l'algorithme de classification binaire majoritaire qui attribue à un nouveau point l'étiquette de la classe la plus représentée dans les exemples d'apprentissage de \mathcal{S} . Intuitivement, cet algorithme s'attend à ce que la classe la mieux représentée sur l'échantillon d'apprentissage soit de fait majoritaire. Est-ce que cet algorithme simple peut n'être qu'équivalent à un algorithme tirant ses prédictions au hasard ? Sans en donner une preuve formelle, il est possible de s'en convaincre intuitivement. En effet, dans les problèmes pour lesquels une classe est nettement majoritaire, on peut s'attendre à ce que dans la plupart des cas l'algorithme majoritaire détecte correctement cette majorité dans l'échantillon d'apprentissage et soit de ce fait meilleur qu'une prédiction au hasard (de performance 1/2) sur les formes \mathbf{x} non vues. Qu'en est-il alors pour

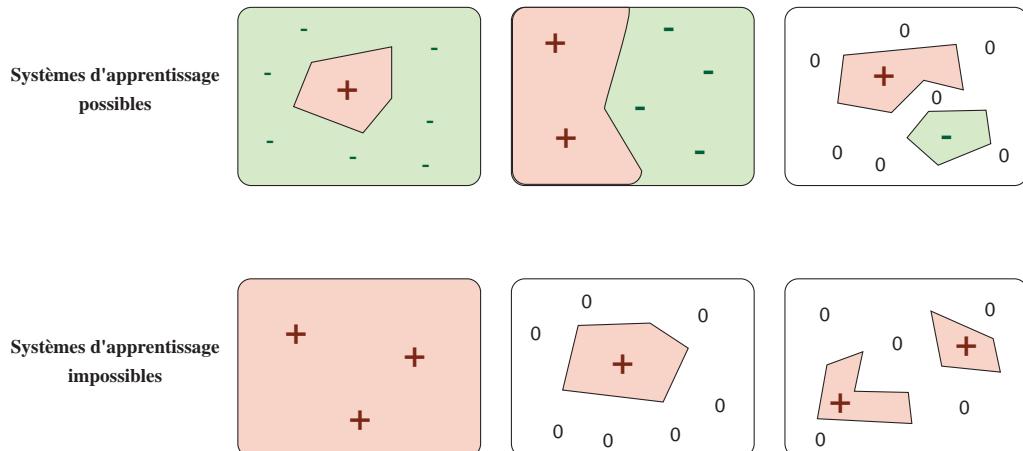


FIG. 17.9 – Le no-free-lunch-theorem prouve que pour chaque région de l'espace des problèmes pour laquelle un algorithme d'apprentissage a une performance supérieure au hasard (indiqué ici par un « + »), il existe une région pour laquelle la performance est moins bonne que le hasard (indiqué ici par un « - »). Un « 0 » indique ici la performance d'un algorithme au hasard, donc la performance moyenne. Les trois figures du dessus correspondent à des situations possibles pour un algorithme d'apprentissage, tandis que les trois figures du dessous correspondent à des situations impossibles : celles d'un algorithme qui serait intrinsèquement supérieur à un algorithme au hasard quand on le considère sur l'ensemble des problèmes possibles (d'après [Sch94]).

les autres problèmes, ceux pour lesquels il n'existe pas de majorité nette, et qui d'après la loi binomiale sont de très loin les plus nombreux ? Est-ce que l'algorithme majoritaire n'est pas sur ceux-là équivalent à un algorithme au hasard, contredisant ainsi le *no-free-lunch theorem* ? Même si les deux classes sont également représentées sur \mathcal{X} , les variations d'échantillonnage feront que souvent l'une d'entre elles sera prévalente dans \mathcal{S} , entraînant une prédiction dans ce sens par l'algorithme majoritaire alors que, sur les exemples non vus, ce sera naturellement l'autre classe qui sera (un peu) mieux représentée. L'algorithme, sur ces problèmes, fera donc (un peu) moins bien que l'algorithme de prédiction au hasard. En prenant en compte tous les cas possibles, la performance globale de cet algorithme ne sera pas meilleure que celle de l'algorithme au hasard. Un raisonnement similaire montre que la courbe de généralisation de l'algorithme majoritaire peut être décroissante. Encore une fois, dans les cas où une classe est clairement majoritaire, l'algorithme majoritaire va avoir de plus en plus de chance de détecter correctement cette majorité avec des tailles d'échantillon croissantes (voir figure 17.10 (a)). Si en revanche les deux classes sont également représentées sur \mathcal{X} , alors la courbe va être décroissante (voir figure 17.10 (b)). En effet, pour les petites tailles d'échantillon, la performance sera seulement légèrement inférieure à $1/2$, puisque lorsque l'algorithme détectera une majorité dans son échantillon, ce sera l'autre classe qui sera de fait mieux représentée sur les exemples restants, mais de très peu. En revanche, plus l'échantillon d'apprentissage est important, plus le choix, forcément mauvais, de l'algorithme entraînera un mauvais taux de prédiction sur les exemples restants. À la limite, quand tous les exemples sauf un auront été vus par l'algorithme d'apprentissage, la prédiction sur le dernier sera forcément mauvaise (la classe prévalente sur \mathcal{S} étant la classe opposée à celle de ce dernier), et la performance tombera à 0.

Quelles leçons tirer de ce théorème ? Faut-il jeter ce livre par terre et se maudire d'avoir consacré déjà tant de temps à étudier une science sans avenir ? Le *no-free-lunch theorem* n'empêche

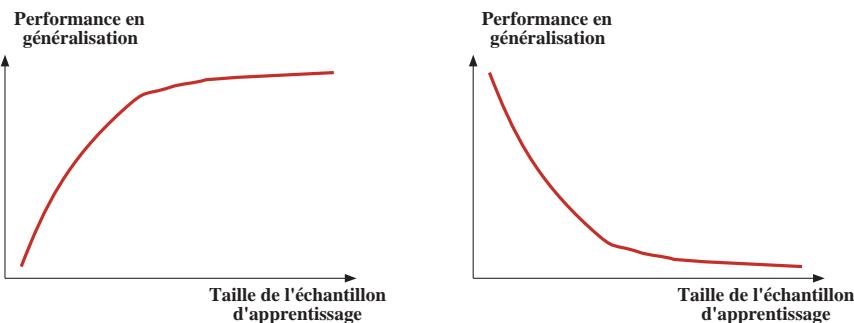


FIG. 17.10 – *Le no-free-lunch theorem prouve que pour chaque région de l'espace des problèmes pour laquelle un algorithme d'apprentissage a une courbe en généralisation croissante (a), il existe une région pour laquelle il existe une courbe en généralisation décroissante, c'est-à-dire indiquant que plus l'algorithme apprend, moins il est performant sur les formes non vues (b). (D'après [Sch94].)*

pas de travailler sur les problèmes inductifs, il averti simplement que la prudence est de rigueur. Plus précisément :

1. Un algorithme d'apprentissage est forcément biaisé vers une certaine classe de problèmes. C'est toujours en fonction de certains *a priori* sur les problèmes à résoudre qu'il faut concevoir et utiliser un algorithme d'apprentissage.
2. Il n'est pas admissible de parler de la performance d'un algorithme sans préciser sur quelle classe de problèmes il a été testé et pour quelle classe de problèmes il a été conçu.
3. L'induction ne crée pas d'information. Elle ne fait que transformer une information *a priori*, inscrite dans les biais de l'algorithme d'apprentissage, et qui est révélée par l'intermédiaire d'un échantillon d'apprentissage. Si l'information *a priori* est inadaptée à la situation rencontrée, le résultat sera également mauvais.

D'un certain côté, le *no-free-lunch theorem* est une nouvelle occasion de ne pas croire aux miracles. Il existe d'ailleurs d'autres versions de ce théorème pour des problèmes importants pour l'apprentissage :

1. Le *théorème du vilain petit canard* [Wat85] dit qu'il n'existe pas *a priori* de meilleur ensemble de descripteurs pour décrire des formes, et qu'en l'absence d'autres informations, il n'existe pas de meilleure notion de similarité entre formes. Toute similarité est dépendante de biais qui peuvent, ou non, être corrects pour l'application étudiée.
2. Le *no-free-lunch theorem pour les algorithmes d'optimisation* [Wol97] dit qu'en moyenne sur tous les problèmes de recherche d'un *extremum* d'une fonction de coût, il n'existe pas d'algorithme de recherche qui soit intrinsèquement meilleur que tout autre algorithme de recherche. Cela signifie en particulier que les algorithmes de recherche par gradient, ou par recuit simulé ou par évolution simulée, tout aussi sophistiqués soient-ils, sont susceptibles d'être pires qu'une recherche au hasard sur certaines classes de problèmes.

Rendu à ce point, le lecteur peut se détendre, commencer à accepter les implications de ces théorèmes de conservation et envisager la suite avec la sérénité qui sied au sage. Pourtant... Pourtant il ne serait pas déraisonnable que certains se réveillent brutalement la nuit en proie à des palpitations et des sueurs froides. Parce que nous les avons précédés dans cette épreuve, nous allons partager ce moment d'inquiétude, violent mais salutaire.

17.4.2 Le *no-free-lunch theorem* et l'analyse de Vapnik : une contradiction ?

Le *no-free-lunch theorem* affirme qu'on ne peut compter sur aucune corrélation entre l'échantillon d'apprentissage \mathcal{S} et les exemples non vus. L'analyse de Vapnik exprime la corrélation entre le risque empirique, mesuré sur \mathcal{S} , et le risque réel. En gros, cette analyse dit que, pour une certaine taille m de l'échantillon d'apprentissage, et pour une certaine richesse de l'espace d'hypothèse caractérisée par exemple par la dimension de Vapnik-Chervonenkis, on peut borner, en probabilité, l'écart entre le risque empirique mesuré et le risque réel. D'un côté, donc, il n'y a pas *a priori* de corrélation entre le passé et le futur, de l'autre, on peut avoir une certaine garantie que la performance passée soit représentative de la performance future. Les deux théorèmes sont corrects. Où est la faille ?

Réexaminons le théorème de Vapnik. Il nous dit qu'étant données une fonction cible f et une hypothèse h tirée d'un espace de fonctions de richesse limitée, il y a très peu de chances, disons moins de 5 %, que sur un échantillon de m points tirés au hasard suivant $d_{\mathcal{X}}$, je ne me rende pas compte que h est de fait éloignée de f (au sens du risque réel). D'après cette interprétation, il semble donc que si je trouve dans un espace d'hypothèses de richesse limitée une hypothèse de risque empirique faible, j'ai de bonnes garanties (par exemple supérieures à 95 %) que le risque réel soit du même ordre. Certes, il reste 5 % de chances que l'échantillon \mathcal{S} n'ait pas été représentatif, au sens où il ne m'aurait pas permis de découvrir la supercherie, c'est-à-dire que h est de fait mauvaise, mais il semble que ce danger soit circonscrit, et le *no-free-lunch theorem* une menace tout compte fait exagérée. Pouvons-nous enfin nous reposer sur notre mol oreiller et dormir apaisé ?

Reconsidérons tout cela une nouvelle fois. Étant donné un échantillon d'apprentissage \mathcal{S} , il existe tout un ensemble de fonctions de risque empirique faible sur cet échantillon. Par exemple, dans le cas de fonctions booléennes définies sur un espace à 12 descripteurs, il existe $2^{2^{12}} = 2^{4096}$ fonctions des $2^{12} = 4096$ formes d'entrées possibles. Supposons que nous ayons un échantillon d'apprentissage donnant la réponse pour 1024 exemples, soit 1/4 de toutes les formes possibles, il reste $2^{2^{12}-1024} = 2^{3072}$ fonctions qui s'accordent aux données d'apprentissage, c'est-à-dire de risque empirique nul. Supposons alors que nous ayions choisi *a priori* un espace d'hypothèses de richesse suffisamment limitée pour que les bornes de Vapnik nous disent qu'il y a 95 % de chances que si le risque empirique mesuré sur un échantillon de taille 1024 est nul, alors le risque réel est inférieur à ε . Devons-nous alors penser que, si nous avons trouvé dans cet espace limité une hypothèse de risque empirique nul, nous avons de grandes chances (supérieure à 95 %) d'avoir identifié une bonne hypothèse (de risque réel $< \varepsilon$) ? Il est clair que non. Nous avons trouvé l'une des 2^{3072} fonctions cohérentes avec les données, et certes il était peu probable *a priori* que ce soit le cas dans notre espace limité d'hypothèses. Cependant la vaste majorité de ces 2^{3072} fonctions a un risque réel $> \varepsilon$ si celui-ci est assez petit. Il est donc très probable que nous soyons tombés sur l'une de ces fonctions apparemment bonnes (sur l'échantillon d'apprentissage), mais effectivement mauvaises (sur le reste des exemples). C'est là où il faut bien comprendre la nature des théorèmes de Vapnik. Ils sont valables en prenant en compte *tous* les échantillons possibles de taille m . Effectivement, par exemple 95 % des échantillons de taille 1024 permettraient de détecter que l'hypothèse choisie est mauvaise. Mais lorsque nous sommes face à un problème donné, nous avons affaire à un échantillon particulier \mathcal{S} . Sur cette base seule, malheureusement, nous ne pouvons avoir aucune certitude, ni même de garantie rassurante.

Nous ne pouvons pas échapper au *no-free-lunch theorem*. L'échantillon d'apprentissage seul ne permet pas d'avoir la moindre garantie sur la performance de l'induction réalisée, il faut avoir d'autres informations sur le problème étudié.

Pour résumer :

- Sans biais, c'est-à-dire sans restriction sur l'espace d'hypothèses considéré, l'induction est impossible.
- Si à cause de connaissances préalables ou par chance on dispose d'un bon biais, alors il y a de fortes chances (mesurées par les théorèmes de Vapnik par exemple) qu'une hypothèse bonne sur les données d'apprentissage, soit bonne en espérance (risque réel).
- Si on ne dispose pas d'un biais adéquat (mauvaises connaissances préalables) :
 1. On a de fortes chances de s'en rendre compte sur l'échantillon d'apprentissage en ne trouvant pas d'hypothèses de risque empirique faible.
 2. Si on trouve une hypothèse de risque empirique faible, on ne peut pas savoir que c'est par erreur.

L'induction n'est vraiment pas faite pour les coeurs faibles ou les foies jaunes !

17.5 Discussion sur l'analyse classique. Variantes et perspectives

Voici donc le terme d'un ouvrage volumineux que beaucoup de spécialistes trouveront trop court, tant il y a de choses qui n'ont pas été dites. La science de l'apprentissage artificiel a en effet été fructueuse. Un cadre théorique s'est vigoureusement développé, solidement enraciné dans les théories statistiques de lois de convergence, la théorie bayésienne et la théorie de la complexité algorithmique. De nombreux algorithmes et techniques d'apprentissage ont été mis au point, que les praticiens de multiples domaines sont avides d'employer (génomique, fouille de données en entreprise, études de marché, etc.). La communauté des chercheurs et des praticiens de l'apprentissage artificiel est active et reconnue institutionnellement : avec des postes dans les universités et les laboratoires, des conférences et des revues spécialisées. Pourtant cette science, si vive, si féconde, répond-elle à toutes les interrogations sur l'apprentissage ?

Lorsque l'on prend du recul, on peut être surpris par l'image de l'apprentissage qu'elle dessine. On s'y intéresse en effet à des agents isolés, recevant passivement des données produites de manière aléatoire par une Nature indifférente. Ces agents ne cherchent pas vraiment à comprendre le monde où ils se trouvent, mais tentent « seulement » d'être bons en moyenne (voir la figure 17.11). D'ailleurs, ils n'évoluent pas. Une seule dose de données ingurgitée d'un seul coup, et c'en est fini pour toujours. La science de l'apprentissage est une science du statique et non du dynamique ! Ce n'est pas plus une science de l'information ou de la connaissance : l'expression des connaissances préalables est très pauvre, se résumant essentiellement à des *a priori* sur les fonctions cible possibles et à l'algorithme utilisé ; les connaissances produites consistent le plus souvent en des procédures de décision, parfois complètement opaques comme dans les réseaux connexionnistes. Il ne s'agit évidemment pas ici de dénoncer les recherches menées en apprentissage artificiel, mais il est clair que le paradigme actuel, par ailleurs si puissant, est notoirement limité. Il y a encore des révolutions scientifiques à mener pour les esprits audacieux. Sans décrire ces révolutions à venir, il est intéressant de voir que certaines directions de recherche actuelles tendent à élargir le cadre dominant.

Sans tout bouleverser, que peut-on remettre en cause dans le cadre classique ?

- **Le critère de performance.** La plupart du temps, ce critère cherche à définir l'écart entre l'état de la Nature, la fonction cible par exemple, et son estimation par l'agent apprenant. Cet écart, qui fonde toutes les approches relevant de la théorie de l'approximation, a deux aspects. D'une part, une mesure de distance ponctuelle, par exemple une distance quadratique entre un point prévu et un point fourni par l'oracle. D'autre part, une densité

de distribution sur l'espace $\mathcal{X} \times \mathcal{U}$ des points. À partir de là, l'écart prend la forme d'une espérance : l'intégrale des distances ponctuelles pondérées par la distribution.

Il faut noter que ce critère n'est pas tourné vers l'identification ou la compréhension de la fonction cible. Il vise en effet à l'efficacité, mais pas à la précision, comme l'illustre la figure 17.11. Intuitivement, il serait en effet intéressant de dédier des ressources de l'apprenant (par exemple des questions à poser, ou bien des paramètres : centres de fonctions noyau, etc.) à l'approximation dans les régions de forte dynamique, mais le critère de performance en espérance conduit à consacrer les ressources aux régions de fortes densités.

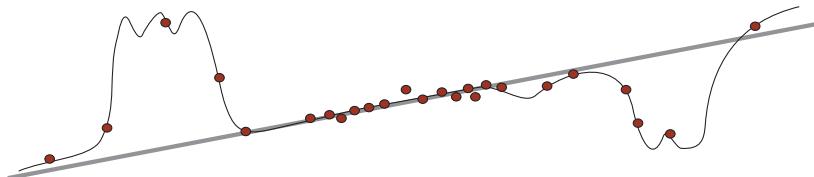


FIG. 17.11 – *Le critère de performance habituel, mesurant une espérance, privilégie l'approximation dans les régions de forte densité de données et non dans les régions de forte dynamique. De ce fait, la fonction identifiée par minimisation du risque empirique, ici une droite, peut être très différente de la fonction cible.*

Mais d'autres critères de performance peuvent être intéressants. Par exemple, la performance d'un système d'apprentissage, l'écart précédemment défini, peut éventuellement varier en fonction d'autres paramètres, comme le taux de faux positifs admis, ou la confiance du système dans sa prédiction. Ce n'est alors plus un nombre, comme le taux d'erreur, qui caractérise un système, mais une courbe, voire une surface. On parle alors d'optimisation multicritère et la comparaison entre systèmes d'apprentissage doit faire intervenir l'ensemble des paramètres.

On peut aussi vouloir prendre en compte la *complexité computationnelle* d'un apprentissage afin de traduire qu'il doit s'effectuer dans un temps raisonnable, compatible avec les exigences de fonctionnement dans le monde. C'est ce que tente de capturer un versant du modèle d'apprentissage PAC (voir chapitre 2) qui pose qu'un apprentissage n'est possible que s'il est de complexité au plus polynomiale en un certain nombre de paramètres⁶. Pour le moment, cette caractérisation formelle des apprentissages réalisables a surtout permis de montrer que certains apprentissages étaient non apprenables dans ce cadre. L'accumulation de ces résultats négatifs a lassé même les théoriciens, et ce d'autant plus qu'ils se fondent sur des bornes souvent grossières que la réalité des applications pratiques semble ignorer. Les théorèmes sur les vitesses de convergence dans l'analyse de Vapnik ont remplacé ce type d'investigations.

Cependant, les critères de performances évoqués ci-dessus privilégiennent le point de vue des systèmes d'apprentissages « à un coup », *batch-learning*, dans lesquels la performance n'est mesurée qu'après l'apprentissage. C'est évidemment très restrictif. La plupart des organismes naturels, les organisations sociales et les institutions, mais aussi certains systèmes artificiels, ne peuvent survivre que s'ils apprennent en permanence et que leur performance tout au long de leur existence est correcte, et pas mesurée seulement une fois à la fin. Il

6. Plus formellement, on dit qu'une classe de concepts \mathcal{F} définie sur un espace d'exemples \mathcal{X} est apprenable avec un espace d'hypothèses \mathcal{H} par un apprenant \mathcal{A} si pour tout $f \in \mathcal{F}$, toute distribution $\mathcal{D}_{\mathcal{X}}$ sur \mathcal{X} , un taux d'erreur ε tel que $0 < \varepsilon < 1/2$ et un taux de confiance δ tel que $0 < \delta < 1/2$, et à partir d'un échantillon d'apprentissage de taille m , l'apprenant \mathcal{A} produit avec une probabilité au moins $(1 - \delta)$ une hypothèse $h \in \mathcal{H}$ telle que $R_{\text{Réel}} \leq \varepsilon$ (où le risque est calculé par un taux d'erreur) en un temps polynomial en $1/\varepsilon$, $1/\delta$, m et $\text{taille}(f)$.

est donc important de définir des mesures de performances qui puissent s'appliquer tout au long de la trajectoire des états suivie par l'apprenant.

Finalement, il faudra bien un jour envisager des mesures de performances plus sophistiquées, prenant en compte à la fois l'intelligibilité des connaissances produites par l'apprenant, mais aussi la manière dont elles peuvent s'inscrire dans les connaissances antérieures, dans les connaissances de la collectivité, humaine ou non, et s'interféconder avec elles. Ce jour-là, l'apprentissage artificiel pourra renouer un dialogue fécond avec d'autres sciences de l'apprentissage, comme la psychologie ou la didactique. Il reste pour cela du chemin à parcourir.

- **Le protocole d'apprentissage.** Il règle le protocole des interactions entre l'apprenant et son environnement, celui-ci incluant éventuellement un oracle ou professeur dans le cas de l'apprentissage supervisé. Nous avons largement examiné les protocoles d'apprentissage supervisé, non supervisé et par renforcement. Ils n'épuisent cependant pas l'ensemble des possibilités et d'autres types d'apprentissages sont envisagés comme l'apprentissage incrémental ou en ligne (*on-line learning*), l'apprentissage actif ou les apprentissages collaboratifs.
- **Le type d'analyse théorique.** Toutes les études théoriques prennent comme base l'hypothèse de données tirées aléatoirement et indépendamment suivant une distribution fixe (tirage i.i.d.). C'est en effet le seul cadre dans lequel on sache établir des théorèmes de convergence uniforme sur des fonctions de distribution. Malheureusement, ou heureusement, l'environnement d'un agent obéit rarement à cette hypothèse. L'agent modifie les distributions de données par son action, la tâche d'apprentissage évolue, la Nature elle-même change. On retombe là sur le problème de la définition d'autres protocoles d'apprentissage et d'autres critères de performance. L'approche théorique de l'apprentissage va devoir regarder ailleurs que dans la théorie statistique.

Dans les sections suivantes, nous revenons brièvement mais de manière un peu plus formelle sur les modèles d'apprentissage définis par des critères de performance et des protocoles différents. Notre but est de permettre au lecteur de situer à quoi se réfèrent certains termes, en lui laissant l'initiative de s'informer plus avant pour ce qui l'intéresse.

17.5.1 D'autres modèles d'apprentissage

17.5.1.1 Apprentissage incrémental ou en ligne

Dans l'apprentissage non incrémental ou encore batch, l'échantillon de données d'apprentissage est fourni d'un seul coup à l'apprenant. Celui-ci peut alors choisir une hypothèse optimisant sur ces données un certain critère objectif traduisant le principe inductif: minimisation d'un risque, compression maximale de l'information ou maximisation d'une vraisemblance, par exemple. Dans l'apprentissage incrémental en revanche, les données sont fournies séquentiellement à l'apprenant et celui-ci doit prendre des décisions et mettre à jour son estimation du monde après chaque nouvelle donnée. La performance n'est donc plus mesurée après l'apprentissage, comme un risque ou une espérance de risque, mais traduit la qualité des décisions prises tout au long de l'apprentissage. Généralement, on suppose qu'à chaque étape (*trial*), l'apprenant reçoit une donnée, prend une décision: par exemple prédit l'étiquette de cette donnée, puis subit un coût qui est fonction de sa prédiction ou de son estimation de l'état du monde, et du véritable état du monde (parfois aussi du changement d'estimation d'une étape à la suivante). L'apprenant met alors à jour son estimation courante du monde et attend la prochaine étape. Comment mesurer la performance globale du système alors que la séquence de données peut ne pas être bornée?

Principalement deux types de scénarios ont été explorés.

- Le premier s'appelle *apprentissage incrémental avec nombre d'erreurs* (*mistake-bound learning model*). Il est assez naturel. On se demande combien d'erreurs peut-on être amené à faire dans le pire des cas (le pire concept cible et la séquence d'exemples la plus désavantageuse) pour identifier un concept cible. Plus formellement, à chaque étape, un exemple $x \in \mathcal{X} = \{0, 1\}^d$ est fourni à l'algorithme qui doit prédire sa classe 0 ou 1 avant de recevoir sa véritable étiquette. L'algorithme est pénalisé pour chaque erreur de prédiction commise. Le but est d'avoir un apprenant faisant le moins d'erreurs possible. On suppose généralement que la présentation des exemples est sous le contrôle d'un adversaire. Dans ce modèle, on s'intéresse particulièrement aux algorithmes qui pour tout concept c dans un espace de concepts cible \mathcal{C} et pour toute séquence d'exemples, ne font pas plus de $\text{poly}(d, \text{taille}(c))$ ⁷ erreurs avec un temps de calcul par étape en $\text{poly}(d, \text{taille}(c))$. On dit alors que l'algorithme apprend \mathcal{C} dans le modèle incrémental avec nombre d'erreurs.
- Le second est le *modèle d'apprentissage incrémental avec perte relative* (*relative loss bound model*). Il se rapporte à toute une lignée de travaux dans des domaines connexes comme l'*apprentissage de stratégies mixtes optimales* dans les jeux itérés, le problème du *codage universel* en théorie de l'information, celui des *portefeuilles universels* en prédiction financière, et plus généralement le problème de la *prédiction universelle*. Dans cette approche, on s'intéresse aux propriétés de la prédiction d'une séquence (et non d'une population de séquences générée par un modèle probabiliste, ce qui mène à des espérances de pertes). La mesure de performance d'un algorithme de prédiction ne peut plus alors se faire dans l'absolu et doit être mesurée par comparaison avec une population \mathcal{F} de prédicteurs, que l'on appelle aussi des *experts*. On cherche alors quelle est la perte ou *regret* maximal de la stratégie d'apprentissage par rapport au meilleur expert de \mathcal{F} . Notons que la notion de regret renvoie aussi à ce qui est perdu par rapport à un apprenant (expert) qui aurait eu toutes les données d'un coup (apprentissage batch). Parfois, on parle d'apprentissage à partir d'avis d'experts (*using expert advices*). Les résultats connus sont encore parcellaires, mais il faut retenir que les pertes calculées dépendent de certaines propriétés métriques de la population d'experts \mathcal{F} .

En dehors du fait que ces travaux prennent en compte le caractère généralement incrémental des apprentissages, l'un de leurs intérêts majeurs est qu'ils peuvent renouveler l'approche théorique de l'apprentissage dans la mesure où l'on s'y affranchit d'hypothèses sur la distribution des exemples et en particulier sur l'hypothèse i.i.d. (distribution indépendante et identique). (Voir [AW01, Blu97, CBFH⁺97, CBL01, Cov91, MF98]).

17.5.1.2 Apprentissage actif et apprentissage guidé

Excepté pour le cas de l'apprentissage par renforcement, cet ouvrage a essentiellement rendu compte de protocoles d'apprentissage dans lesquels l'apprenant est passif, recevant les données que veut bien lui fournir la Nature ou l'oracle. La différence est notable avec les agents cognitifs naturels qui agissent sur le monde et sont en partie responsables du flot de données leur parvenant. Pourquoi alors ne pas étudier ces apprentissages actifs, ne serait-ce que pour savoir s'ils sont avantageux par rapport aux apprentissages passifs ?

La base des modèles proposés dans ce cadre est de supposer que l'apprenant peut poser des questions à la Nature sous des formes diverses.

- Dans le protocole de requête d'appartenance (*membership query*), l'apprenant peut choisir

7. C'est-à-dire une fonction polynomiale de d et de la taille de description du concept c .

une forme et demander quelle est son étiquette à l'oracle. Il s'agit alors de voir en combien de questions au minimum l'apprenant peut identifier la meilleure hypothèse. ([Ang88])

- Dans le protocole de requête d'équivalence (*equivalence query*), l'apprenant peut proposer une hypothèse h , et l'oracle, soit l'informe que l'hypothèse est logiquement équivalente à la fonction cible, soit lui fournit un contre-exemple infirmant l'hypothèse. ([Ang88])
- Dans le protocole de requête statistique (*statistical query model*), l'apprenant ne peut avoir accès directement aux exemples étiquetés, mais peut poser des questions sur les statistiques des exemples étiquetés (par exemple 3/4 des 52 exemples sont positifs). Ce modèle est particulièrement utile dans le cas de données dont l'étiquette peut être erronée (bruit de classification). ([KV94])

Pour chacun de ces protocoles, des résultats ont été obtenus sur des classes de fonctions cible apprenables efficacement dans le modèle PAC et sur des équivalences entre ces modèles. S'il est acquis que certains protocoles permettent de réduire la taille de l'échantillon d'apprentissage nécessaire, aucun résultat ne permet d'affirmer que l'apprentissage actif soit plus puissant que l'apprentissage passif en termes de fonctions apprenables. À l'opposé des apprentissages dans lesquels l'apprenant a l'initiative des questions, existent les modèles d'apprentissage dans lesquels un professeur essaie de faciliter la vie de l'apprenant en choisissant bien les exemples fournis et éventuellement l'ordre dans lequel ils sont présentés. On parle alors d'*apprentissage guidé* (*teachability*). Si ces modèles sont potentiellement très intéressants car ils pourraient donner des informations précieuses sur la manière d'enseigner, ils butent pour le moment sur une définition adéquate de ce qu'est un guidage licite. Si, en effet, on ne prend pas de précaution, il est facile pour l'enseignant d'être complice de l'apprenant et de tricher en codant la fonction cible sous la forme d'un échantillon d'apprentissage. La frontière entre connivence illicite et aide bienveillante est subtile. Les travaux dans ce domaine restent préliminaires [GK95].

17.5.1.3 Apprentissage multi-instance

Il se peut que l'on veuille apprendre à reconnaître des « objets » ayant certaines propriétés quand ces propriétés sont dues à certaines conformations ou manifestations de ces objets, mais pas nécessairement à toutes. Ainsi, par exemple, on voudra apprendre que telle molécule dont on connaît la formule brute a un caractère cancérogène parce qu'au moins l'une de ses conformations est cancérogène. De même, un trousseau de clé est utile parce qu'il contient une clé au moins permettant d'ouvrir la porte. La difficulté de ce type d'apprentissage vient du fait que l'apprenant ne sait pas quelle conformation est responsable de l'étiquette de l'objet et qu'il doit cependant apprendre à prédire ces étiquettes.

Ce type d'apprentissage a été décrit par [DLLP97] dans le cadre de la reconnaissance de molécules cancérogènes. Il correspond à bien d'autres situations pratiques. Conceptuellement, cet apprentissage est intéressant à étudier car il est associé à des langages de représentation dont le pouvoir expressif doit être intermédiaire entre celui de la logique des propositions et celui de la logique des prédictats. Il est ainsi envisageable d'échapper aux limites de l'un et aux problèmes de calculabilité de l'autre. La thèse de Yann Chevaleyre ([Che01]) est une bonne introduction à l'apprentissage multi-instance (*multi-instance learning*).

17.5.2 D'autres types d'analyses

17.5.2.1 Analyse de la physique statistique

L'analyse de Vapnik de l'induction par minimisation du risque empirique (ERM) à partir d'un échantillon aléatoire de m exemples conduit à des courbes d'erreur en généralisation bornées

par $O(d_{\mathcal{H}}/m)$ (dans le cas de la discrimination et d'une fonction cible appartenant à l'espace d'hypothèses \mathcal{H} de dimension de Vapnik-Chervonenkis $d_{\mathcal{H}}$) ou par $O(d_{\mathcal{H}}/m)$ (dans le cas de la discrimination et d'une fonction cible n'appartenant pas à \mathcal{H}). Rappelons que ces bornes, obtenues dans le cadre d'une analyse dans le pire cas, sont universelles : elles sont valables pour tout espace \mathcal{H} d'hypothèses, pour toute distribution de données et pour toute fonction cible. Par ailleurs, il a été montré que ces bornes sont essentiellement les meilleures possibles dans le pire cas, dans le sens où, pour tout espace \mathcal{H} , il existe une distribution de données sur \mathcal{X} pour laquelle la borne inférieure sur l'erreur de généralisation est égale à la borne supérieure donnée ci-dessus. Dès lors, on pourrait croire que le comportement réel en généralisation des algorithmes d'induction est décrit soit par la forme fonctionnelle $d_{\mathcal{H}}/m$ soit par $\sqrt{d_{\mathcal{H}}/m}$. Il se trouve que l'on peut observer toute une variété de comportements ne correspondant pas à ces formes fonctionnelles. Ainsi, bien souvent, des erreurs en généralisation (risques réels) faibles sont obtenues pour des échantillons d'apprentissage beaucoup plus faibles que ceux prédis par la théorie (parfois même pour des échantillons de taille $< d_{\mathcal{H}}$, c'est-à-dire pour lesquels aucune borne en généralisation n'est valide en théorie). Parfois aussi, on observe des courbes d'apprentissage présentant de brutales transitions (voir la figure 17.12) réminiscentes des phénomènes de transitions de phase en physique.

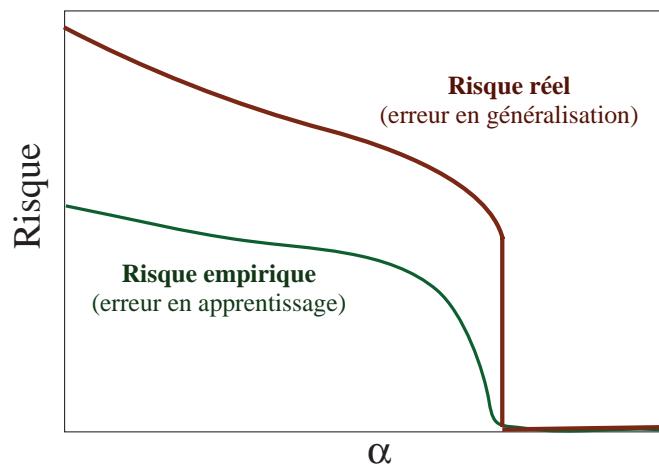


FIG. 17.12 – Un exemple de courbe d'apprentissage présentant une transition brutale vers une erreur en généralisation nulle. On étudie donc les caractéristiques de l'espérance de risque en fonction du rapport $\alpha = m/N$, m étant le nombre d'exemples dans l'échantillon d'apprentissage et N le nombre de degrés de liberté gouvernant \mathcal{H} (typiquement, le nombre de connexions dans un réseau de neurones, sans qu'il soit établi que cela constitue des degrés de liberté indépendants).

Plusieurs explications de ces phénomènes ont été proposées. Par exemple :

- L'algorithme d'apprentissage (e.g. un réseau connexionniste) n'accéderait effectivement qu'à un sous-espace de l'espace des hypothèses \mathcal{H} , dont la dimension de Vapnik-Chervonenkis serait inférieure à $d_{\mathcal{H}}$. Il faudrait donc prendre en compte la dimension de Vapnik-Chervonenkis du sous-espace réellement exploré. Il y a là d'ailleurs une direction de recherche intéressante visant à prendre en compte la stratégie d'exploration de l'algorithme.
- La distribution des données serait en général beaucoup plus favorable que la distribution la pire possible. Cela a motivé des travaux dédiés à l'étude de distributions particulières. Surtout, c'est là l'une des sources de l'excitation autour des séparateurs à vastes marges

(voir chapitre 9 et plus généralement des méthodes à base de fonctions noyau.

Afin d'analyser ces phénomènes dont il semble difficile de rendre compte par l'approche de Vapnik, certains théoriciens ont eu recours à des méthodes issues de la physique statistique. L'idée essentielle est de considérer l'espace \mathcal{H} des hypothèses comme un ensemble d'états possibles d'un système (physique) soumis à la contrainte d'une séquence d'apprentissage donnée. À chaque état (hypothèse) peut être associée une énergie (risque empirique). On cherche alors quelles sont les caractéristiques de cet espace d'états, et, en particulier, pour chaque état possible, la probabilité de s'y retrouver après une exploration stochastique guidée par l'énergie. Ainsi, au lieu d'étudier le risque réel associé à la pire hypothèse minimisant le risque empirique, comme dans l'approche en pire cas, on étudie l'espérance du risque réel dans un espace d'hypothèses sous une distribution de probabilité reflétant la performance en apprentissage de chaque hypothèse. Il s'agit donc bien d'une analyse du principe inductif ERM, mais d'une *analyse en cas moyen* sur l'ensemble de l'espace d'hypothèses en supposant donnés un échantillon d'apprentissage \mathcal{S} et une densité de probabilité *a priori* sur \mathcal{H} .

L'étude de l'espérance de risque réel (équation (17.28))

$$\mathbf{E}_{\mathcal{H}}(R_{R\acute{e}el}(h)) = \int_{h \in \mathcal{H}} R_{R\acute{e}el}(h) p(h|\mathcal{S}) dh \quad (17.28)$$

conduit à examiner la distribution de Gibbs sur l'espace des hypothèses et son évolution en fonction de l'échantillon d'apprentissage \mathcal{S} . Cette grandeur dépend de l'échantillon d'apprentissage \mathcal{S}_m (ce que les physiciens associent à un « désordre gelé » pour indiquer que le système a évolué sous la contrainte fixée posée par \mathcal{S}_m). Il est intéressant de chercher à s'affranchir de cette dépendance en étudiant l'espérance du risque réel moyennée sur tous les échantillons d'apprentissage :

$$\mathbf{E}_{\mathcal{S}_m}[\mathbf{E}_{\mathcal{H}}(R_{R\acute{e}el}(h))] = -\frac{\partial}{\partial \beta} \{\mathbf{E}_{\mathcal{S}_m} \ln [Z_m(\beta)]\} \quad (17.29)$$

Le problème est que le calcul de cette grandeur est en général très difficile. Il n'est résolu que pour des cas particuliers par l'emploi de méthodes encore mal maîtrisées. Deux idées sont essentielles pour aborder ce calcul :

1. Ce qui est important, c'est une sorte de capacité associée à chaque degré de liberté de l'espace d'hypothèses. On étudie donc les caractéristiques de l'espérance de risque en fonction du rapport $\alpha = m/N$, m étant le nombre d'exemples dans l'échantillon d'apprentissage et N le nombre de degrés de liberté gouvernant \mathcal{H} (typiquement, le nombre de connexions dans un réseau de neurones, sans qu'il soit établi que cela constitue des degrés de liberté indépendants). Lorsque l'on fait tendre $m \rightarrow \infty$ en gardant α constant, on parle alors de *limite thermodynamique*. Les courbes d'apprentissage sont établies en examinant l'espérance de risque en fonction du rapport α .
2. On espère que, comme en physique des verres de spin, les propriétés macroscopiques des systèmes d'apprentissage (par exemple leur risque réel) présentent des propriétés d'*automoyennage*. Cela signifie que lorsque les contraintes (l'échantillon d'apprentissage) sont engendrées par une même distribution, les propriétés macroscopiques qui en découlent sont les mêmes et ne dépendent donc pas de la réalisation particulière d'un échantillon d'apprentissage. Cela signifie, qu'à la limite de $N \rightarrow \infty$, tous les échantillons d'apprentissage sont équivalents et l'on peut alors obtenir facilement des propriétés génériques des systèmes d'apprentissage.

Du fait de la difficulté technique des méthodes de calcul mises en jeu et de leurs domaines de validité souvent restreints quand ils ne sont pas incertains, les résultats obtenus sont parcellaires. Nous ne rentrerons pas dans leur détails ici.

L'approche de la physique statistique qui cherche à étudier des propriétés typiques du principe ERM plutôt que des bornes de confiance est potentiellement très intéressante, et ce d'autant plus qu'on peut également obtenir par ce biais des informations sur la dynamique de l'apprentissage et non seulement sur ses propriétés asymptotiques. C'est pourquoi nous croyons utile de l'évoquer dans cet ouvrage.

Cependant, cette approche qui repose sur la mise à jour de propriétés d'automoyennage dans les systèmes d'apprentissage, pose des problèmes redoutables et implique la mise en œuvre de techniques difficiles et dont les domaines de validité sont encore imprécisément connus. Cela explique sans doute le petit nombre de publications la concernant. Sans être exhaustifs, nous pouvons citer en particulier [Gar88, HKS94, HKST96, LTS89, OH91, SST92, WRB93].

17.5.2.2 Apprentissage et analyse des systèmes dynamiques

Un apprenant est un système caractérisé par un certain état qui évolue en fonction de cet état et des entrées dues à l'environnement. On peut donc le caractériser comme un système dynamique. Lorsque l'apprenant est soumis à une séquence d'entrées, il suit une trajectoire le faisant passer d'un état d'origine e_0 à un état final e_f . On peut alors chercher ce qui caractérise ces trajectoires. La physique des systèmes dynamiques nous apprend que la trajectoire suivie par un système rend extrémale une quantité que l'on appelle *action* et qui est l'intégrale le long de la trajectoire d'une quantité appelée Lagrangien (à ne pas confondre avec les multiplicateurs de Lagrange). Si l'on connaît le Lagrangien d'un système, on peut calculer sa trajectoire pour toute séquence d'apprentissage.

Ce qui est intéressant, c'est que cette approche permet de relier la notion d'information avec celle d'apprentissage. En effet, considérons maintenant un apprenant tel que, étant donné un état initial e_0 , quel que soit l'ordre dans lequel est présenté un échantillon d'apprentissage, il parvienne au même état final e_f . En d'autres termes, l'apprenant est insensible à l'ordre de présentation des données. Cela correspond à un invariant sur la trajectoire qui est lié à un invariant de l'action et du Lagrangien. Cette invariance implique des relations spécifiques entre information et prise en compte de cette information par l'apprenant. Notamment, le système ne peut oublier n'importe comment l'information qui lui a été fournie. On peut ainsi établir un lien entre information, système d'apprentissage et oubli. Après tout, il est curieux que la notion d'oubli n'apparaisse qu'ici dans un livre sur l'apprentissage. Apprentissage - oubli / oubli - apprentissage, l'un est-il pourtant dissociable de l'autre?

Il y a encore tellement de choses à apprendre sur l'apprentissage!

Notes historiques et bibliographiques

L'historique de l'analyse de l'apprentissage par Vapnik a déjà été abordée dans les chapitres 2 et 9. Nous nous intéressons donc ici aux autres sujets de ce chapitre.

L'idée que l'induction pouvait être vue comme l'approximation d'une fonction multi-variables régulière à partir de données n'est pas nouvelle. Sa formalisation s'est cependant faite progressivement et c'est vraiment Girosi et Poggio qui se sont faits les champions de ce point de vue en essayant de montrer que toutes les autres approches théoriques peuvent s'y ramener (voir [GJP95] par exemple). Ils ont notamment étudié les propriétés d'un certain nombre de critère de pénalisation ainsi que des modèles dans lesquels les variables d'entrées sont d'abord prétraitées

par des fonctions de base, dont les fonctions à base radiale. Le cours de Girosi au MIT est à cet égard intéressant à consulter.

La théorie de l'estimation bayésienne est bien présentée dans [DHS01, Bis95, CL96] avec des détails historiques dans la première référence. La difficulté de sa mise en œuvre lui fait préférer des versions simplifiées (voir les chapitres 2 et 14).

Les liens entre induction et économie d'expression d'un modèle sont très anciens comme le montre le principe du rasoir d'Occam. C'est Solomonoff [Sol64] qui le premier en 1963 exposa une théorie de l'induction basée sur l'idée d'utilisation d'une probabilité *a priori* liée à la complexité de Kolmogorov. Le principe de longueur de description minimale (*MDL*) a été introduit indépendamment par Wallace et Boulton [WB85] d'une part, et par Rissanen [Ris78] d'autre part. De nombreux travaux de nature plutôt empirique ont cherché à en tester le champ d'application. Par ailleurs, les débats théoriques actuels portent sur les liens entre le MDL et la théorie bayésienne: celui-ci est-il premier par rapport à celle-ci? (voir les passionnantes débats sur ce sujet à NIPS-2001 (à paraître chez MIT Press)).

Le *no-free-lunch theorem* a des antécédents dans le « théorème du vilain petit canard » [Wat85] énoncé en 1963 à propos de la non-universalité de toute mesure de distance. Sa description et sa preuve sont dues à Wolpert [Wol92] et [Wol95], de même que sa version pour les méthodes d'optimisation [Wol97]. Ce théorème a fait couler beaucoup d'encre dans les années 1990, mais il semble maintenant accepté par la communauté. À notre connaissance, la confrontation avec l'analyse de Vapnik est exposée ici pour la première fois.

Résumé

Dès que l'on aborde des domaines complexes, il faut faciliter les raisonnements et l'apprentissage :

- en contrôlant l'expression de l'espace des hypothèses, par exemple en réalisant des abstractions ;
- en apprenant des connaissances permettant une exploration plus efficace des hypothèses, par exemple à l'aide de macro-opérateurs ou d'heuristiques de contrôle que rend possible l'apprentissage à partir d'explications ;
- en facilitant le transfert de connaissances et de solutions entre domaines, par exemple en utilisant des raisonnements comme l'analogie.

Toutes ces techniques requièrent une ou des théories du domaine fortes. C'est de là qu'elles tirent une grande puissance en permettant l'apprentissage à partir de peu de données. C'est là aussi la source des difficultés de leur application.

Chapitre 18

Annexes techniques

18.1 Exemples de fonctions de perte en induction

Bien que les problèmes d'apprentissage liés à la classification, la régression ou l'estimation de densité soient apparemment très différents, impliquant des espaces d'entrée et de sortie de nature diverse, ils peuvent cependant être analysés à l'intérieur du même cadre d'un problème d'optimisation du risque réel. Il suffit pour cela d'introduire des fonctions de perte adaptées à chaque cas. Cette annexe présente certaines d'entre elles.

18.1.1 La reconnaissance de formes ou classification

On appelle problème de *discrimination*, ou d'*apprentissage de concept*, un problème d'apprentissage de règle de classification pour lequel l'espace de sortie est binaire: $\mathcal{U} = \{0, 1\}$. Il y a donc seulement deux classes possibles: l'une vérifiant le concept à apprendre et l'autre définissant son opposé. L'espace de sortie de la machine \mathcal{Y} n'a donc besoin de prendre que deux valeurs et l'espace \mathcal{H} des fonctions hypothèse est alors celui des *fonctions indicatrices*, prenant leur valeur dans $\{0, 1\}$.

Il est alors courant de prendre une fonction de perte qui mesure l'erreur de classification pour chaque forme présentée à l'apprenant:

$$l(\mathbf{u}_i, h(\mathbf{x}_i)) = \begin{cases} 0 & \text{si } \mathbf{u}_i = h(\mathbf{x}_i) \\ 1 & \text{si } \mathbf{u}_i \neq h(\mathbf{x}_i) \end{cases} \quad (18.1)$$

Avec cette fonction de perte, le risque

$$R_{R\acute{e}el}(h) = \int_{\mathcal{Z}=\mathbf{x} \times \mathbf{u}} l(\mathbf{u}, h(\mathbf{x})) dF(\mathbf{x}, \mathbf{u})$$

mesure la *probabilité de mauvaise classification* (en anglais, *mis-classification*).

Le problème de discrimination est donc celui de l'apprentissage d'une fonction indicatrice minimisant la probabilité d'erreur lorsque la distribution des formes $F(x, u)$ est inconnue et que seul est fourni un échantillon de données.

Il est important de noter que ce type de fonction de perte n'est pas forcément celui qui doit être employé pour tous les problèmes de classification. Pour reprendre l'exemple du diagnostic de l'appendicite, il est beaucoup plus coûteux socialement et financièrement de passer à côté d'une appendicite que d'en diagnostiquer une à tort. Il faut donc dans ce cas définir une fonction de perte qui rende compte de cette asymétrie. De même, les problèmes de classification impliquant plus que deux classes appellent d'autres fonctions de perte.

18.1.2 La régression

La régression consiste à estimer une fonction f à valeurs réelles, connaissant un échantillon fini de couples $(\mathbf{x}, \mathbf{u} = f(\mathbf{x}))$ ou $(\mathbf{x}, \mathbf{u} = f(\mathbf{x}) + \text{bruit})$.

La fonction f à estimer peut donc être considérée comme la somme d'une fonction déterministe et d'un signal d'erreur aléatoire de moyenne nulle (et le plus souvent considéré comme une gaussienne).

$$\mathbf{u} = f(\mathbf{x}) + \epsilon \quad (18.2)$$

On peut aussi décrire ce phénomène en considérant que la fonction déterministe est la moyenne de la probabilité conditionnelle sur l'espace de sortie \mathcal{U} .

$$f(\mathbf{x}) = \int \mathbf{u} p(\mathbf{u}|\mathbf{x}) d\mathbf{u} \quad (18.3)$$

L'espace des fonctions hypothèse \mathcal{H} de l'apprenant peut ou non inclure l'espace des fonctions cible \mathcal{F} . Une fonction de perte usuelle pour la régression est la fonction erreur quadratique (L_2):

$$L(\mathbf{u}_i, h(\mathbf{x}_i)) = (\mathbf{u}_i - h(\mathbf{x}_i))^2 \quad (18.4)$$

L'apprentissage consiste alors à trouver la fonction $h \in \mathcal{H}$ minimisant la fonctionnelle de risque:

$$R_{R\acute{e}el}(h) = \int_{\mathcal{Z}=\mathcal{X} \times \mathcal{U}} (\mathbf{u} - h(\mathbf{x}))^2 dF(\mathbf{x}, \mathbf{u}) \quad (18.5)$$

sur la seule base de l'échantillon d'apprentissage. Cette fonctionnelle, le *risque réel*, mesure la précision des prédictions de l'apprenant.

Remarque

Sous l'hypothèse que le signal d'erreur est une gaussienne centrée en 0, ce risque peut aussi être écrit en fonction de la capacité de l'apprenant à approximer la fonction cible $f(x)$ (et non la sortie \mathbf{u}), comme le montre le calcul suivant :

$$\begin{aligned} R_{R\acute{e}el}(h) &= \int (\mathbf{u} - f(\mathbf{x}) + f(\mathbf{x}) - h(\mathbf{x}))^2 p(\mathbf{x}, \mathbf{u}) d\mathbf{x} d\mathbf{u} \\ &= \int (\mathbf{u} - f(\mathbf{x}))^2 d\mathbf{x} d\mathbf{u} + \int (h(\mathbf{x}) - f(\mathbf{x}))^2 p(\mathbf{x}) d\mathbf{x} \\ &\quad + 2 \int (\mathbf{u} - f(\mathbf{x}))(f(\mathbf{x}) - h(\mathbf{x})) p(\mathbf{x}, \mathbf{u}) d\mathbf{x} d\mathbf{u} \end{aligned} \quad (18.6)$$

Sous l'hypothèse que le bruit est de moyenne nulle, le dernier terme dans la somme ci-dessus s'écrit :

$$\begin{aligned} \int (\mathbf{u} - f(\mathbf{x}))(f(\mathbf{x}) - h(\mathbf{x})) p(\mathbf{x}, \mathbf{u}) d\mathbf{x} d\mathbf{u} &= \int (\epsilon(f(\mathbf{x}) - h(\mathbf{x})) p(\mathbf{u}|\mathbf{x}) p(\mathbf{x}) d\mathbf{x} d\mathbf{u} \\ &= \int (f(\mathbf{x}) - h(\mathbf{x})) \left[\int \epsilon p(\mathbf{u}|\mathbf{x}) d\mathbf{u} \right] p(\mathbf{x}) d\mathbf{x} \\ &= \int (f(\mathbf{x}) - h(\mathbf{x})) E - \epsilon(\epsilon|\mathbf{x}) p(\mathbf{x}) d\mathbf{x} = 0 \end{aligned} \quad (18.7)$$

Le risque peut donc être récrit comme :

$$R_{R\acute{e}el}(h) = \int (\mathbf{u} - f(\mathbf{x}))^2 p(\mathbf{x}, \mathbf{u}) d\mathbf{x} d\mathbf{u} + \int (h(\mathbf{x}) - f(\mathbf{x}))^2 p(\mathbf{x}) d\mathbf{x} \quad (18.8)$$

Le premier terme ne dépend pas de la fonction d'approximation h et peut être écrit en terme de la variance sur le bruit :

$$\begin{aligned} \int (\mathbf{u} - f(\mathbf{x}))^2 p(\mathbf{x}, \mathbf{u}) d\mathbf{x} d\mathbf{u} &= \int \epsilon^2 p(\mathbf{u}|\mathbf{x}) p(\mathbf{x}) d\mathbf{x} d\mathbf{u} \\ &= \int \left[\int \epsilon^2 p(\mathbf{u}|\mathbf{x}) d\mathbf{u} \right] p(\mathbf{x}) d\mathbf{x} \\ &= \int E_\epsilon(\epsilon^2|\mathbf{x}) p(\mathbf{x}) d\mathbf{x} \end{aligned} \quad (18.9)$$

Et substituant (18.9) dans (18.8), on obtient :

$$R_{Réel}(h) = \int E_\epsilon(\epsilon^2|\mathbf{x}) p(\mathbf{x}) d\mathbf{x} + \int (h(\mathbf{x}) - f(\mathbf{x}))^2 p(\mathbf{x}) d\mathbf{x} \quad (18.10)$$

Le risque pour le problème de régression (sous l'hypothèse de fonction de perte L_2 et de bruit de moyenne nulle) a donc une contribution exprimant la variance du bruit et une contribution exprimant la précision de la fonction d'approximation. Comme la variance du bruit est indépendante de la fonction d'approximation, la minimisation du second terme de (18.10) est équivalente à la minimisation du risque (18.5). Ainsi, chercher à obtenir le risque de prédiction optimal est équivalent chercher l'approximation la plus précise de la fonction cible inconnue f .

18.1.3 L'estimation de densité

Un autre problème inductif important consiste à estimer une densité de probabilité dans l'espace d'entrée \mathcal{X} à partir d'un échantillon de données $\{x_i\}_{1 \leq i \leq m}$. Dans ce cas, il n'y a pas nécessité de considérer un espace de sortie, et la sortie $h(\mathbf{x})$ de l'apprenant représente une densité sur \mathcal{X} .

La fonction de perte usuelle dans ce cas est la fonction :

$$l(h(\mathbf{x})) = -\ln h(\mathbf{x}) \quad (18.11)$$

donnant la fonctionnelle de risque :

$$R_{Réel}(h) = \int -\ln h(\mathbf{x}) p(\mathbf{x}) d\mathbf{x} \quad (18.12)$$

Il est établi[DHS01] que la densité optimale h^* minimise cette fonctionnelle de risque. Par ailleurs, si la densité cible $f \notin \mathcal{H}$, alors on peut montrer que la solution h minimisant l'espérance de risque ou risque réel est caractérisable : c'est celle dont la *divergence de Kullback-Leibler* avec la vraie densité f est la plus faible. (Voir la définition de la divergence de Kullback-Leibler dans le chapitre 2).

18.2 Optimisation par descente de gradient

Introduction

Soit une fonction réelle $f(u)$, avec $u \in D \subset \mathbb{R}^d$. On cherche à trouver la valeur u^* du vecteur u telle que :

$$f(u^*) = \underset{u \in D}{\operatorname{ArgMin}} f(u)$$

Autrement dit, on cherche une valeur u^* pour laquelle la fonction f prend la valeur minimale sur son domaine.

Les méthodes d'optimisation itérative consistent, d'une manière générale, à partir d'une valeur u^0 , puis à construire une suite de valeurs u^n telles que :

$$f(u^{n+1}) \leq f(u^n)$$

La construction d'une telle suite permettra de trouver u^* si F remplit des conditions que nous allons énoncer au fur et à mesure.

Définition

Notons $\frac{\partial}{\partial u}(u^n)$ la valeur du gradient de f au point u^n . Rappelons que cette valeur est un vecteur de \mathbb{R}^d .

Les techniques de *descente de gradient* sont des méthodes d'optimisation itérative pour lesquelles on choisit :

$$u^{n+1} = u^n - \alpha_n \frac{\partial}{\partial u}(u^n) \quad (18.13)$$

avec α_n réel positif.

En particulier, si l'on choisit α_n de sorte que :

$$f(u^{n+1}) = \inf_{\alpha} \left(u^n - \alpha \frac{\partial}{\partial u}(u^n) \right) \quad (18.14)$$

la méthode porte le nom de gradient à *descente maximale*.

Une visualisation¹. intuitive de cette technique peut être proposée ainsi: un randonneur dans le brouillard cherche à redescendre le plus vite possible de la montagne où il est perdu. Il observe les alentours et choisit de prendre la direction de la plus grande pente. Si la pente est faible dans cette direction, il sera obligé de recommencer l'opération à peu de distance: le relief est en effet difficile à deviner dans la zone où il se trouve. Si la pente est forte et la montagne assez régulière, il peut par contre raisonnablement penser que le profil du terrain ne va pas se modifier immédiatement; il peut parcourir plus de chemin dans la direction choisie

1. Une autre métaphore est celle du joueur de golf: quand il est loin de la cible, il utilise un club « long » et imprécis. Quand il se rapproche, il réduit la longueur de son coup pour gagner en précision. Il y a aussi des cas de divergence.

Le cas idéal

On dit qu'une fonction $f : D \rightarrow \mathbb{R}$ est α -convexe, si, par définition, il existe un réel positif α tel que :

$$\forall u \in D, \forall v \in D, \forall \delta \in \{0, 1\} : f((1 - \delta)u + \delta v) \leq (1 - \delta)f(u) + \delta f(v) + \frac{\alpha}{2}\delta(1 - \delta)\|u - v\|^2$$

Supposons f différentiable et α -convexe. En prenant pour α_n la solution de l'équation 18.14 et en notant $\langle\langle \cdot, \cdot \rangle\rangle$ le produit scalaire, on a :

$$\langle\langle \frac{\partial}{\partial u}(u^n - \alpha_n \frac{\partial}{\partial u}(u^n)), \frac{\partial}{\partial u}(u^n) \rangle\rangle = 0$$

soit :

$$\langle\langle \frac{\partial}{\partial u}(u^{n+1}), \frac{\partial}{\partial u}(u^n) \rangle\rangle = 0$$

Ce qui signifie que chaque pas de descente maximale se fait dans une direction orthogonale au précédent. On peut alors démontrer que :

$$\lim_{n \rightarrow +\infty} (\|u^{n+1} - u^n\|) = 0$$

La suite u_n tend donc vers une limite. De plus, cette limite est la valeur cherchée u^* , qui minimise f .

N.B. : la démonstration demande au passage que la fonction f soit lipschitzienne, c'est-à-dire que

$$\forall u \in D, \forall v \in D, \exists c \in \mathbb{R} : \|f(u) - f(v)\| \leq c\|u - v\|$$

Par conséquent, si les propriétés de f sont assez fortes, la descente de gradient amène au point cherché.

On peut aussi démontrer que la convergence est encore vraie pour α_n fixé à une valeur positive bornée par une constante dépendant de α et de c .

Le cas général : quelques problèmes

La propriété d' α -convexité est très forte. Intuitivement, elle signifie que le graphe de la fonction f a une courbure supérieure en tout point à la « parabole » $g(v) = 1/2\alpha\|v\|^2$.

La propriété d'être lipschitzienne est moins exigeante : elle implique une régularité qui n'est pas rare en pratique.

Quand on cherche à minimiser une fonction, on ignore en général si elle vérifie les deux propriétés précédentes. La plupart du temps, on peut au mieux la supposer dérivable. Que donne alors la méthode de la descente de gradient ?

La figure 18.1, donne une intuition de la méthode et de ses problèmes. Elle représente le graphe d'une fonction dérivable sur le domaine $D = [0, 1]$. On y voit trois cas caractéristiques :

- Si u_n se trouve dans la partie droite du graphe, le gradient $\frac{\partial}{\partial u}(u^n)$ est positif et grand. Par conséquent, en prenant

$$u^{n+1} = u^n - \alpha \frac{\partial}{\partial u}(u^n)$$

avec α réel positif, on obtiendra par exemple u^{n+1} au point indiqué. À cet endroit, le gradient y est plus petit, quoique toujours positif. Le point suivant u^{n+2} , calculé avec la même valeur de α , sera donc plus proche de u^{n+1} que u^{n+1} l'était de u_n . On assiste de la sorte à la convergence de la suite vers la valeur minimale u^* cherchée.

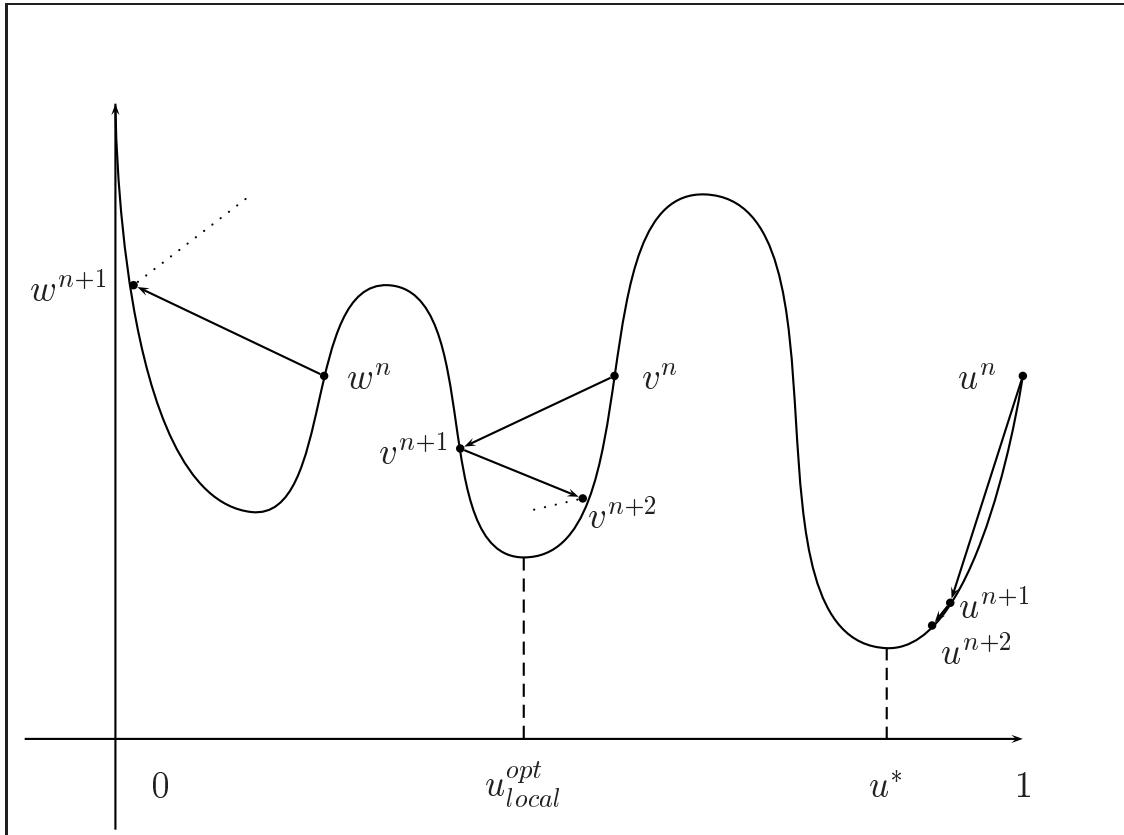


FIG. 18.1 – Une illustration de la méthode de descente de gradient.

- Construisons maintenant une autre suite à partir du point v^n , avec une valeur supérieure pour α . Cette suite converge aussi, mais il y a deux différences avec le cas précédent :
 - la suite n'est plus strictement croissante ou décroissante, mais converge par valeurs alternativement supérieures et inférieures à la valeur finale;
 - la valeur atteinte n'est pas le minimum cherché, mais un *minimum local* u_{local}^{opt} . Cette « erreur » de convergence est le résultat de l'affaiblissement des propriétés de f . Dans la pratique, ce cas se rencontre très souvent.
- À partir du point w^n , en prenant α encore plus grand, par contre la suite va diverger.

En pratique

Rappelons que dans le cas général u est un vecteur. En général, son domaine D sera convexe, par exemple défini par un hyperrectangle :

$$D = \{l_1^{min} \leq x_1 \leq l_1^{max}, \dots, l_d^{min} \leq x_d \leq l_d^{max}\}$$

De nombreuses méthodes ont été proposées pour pallier les faiblesses présentées sur l'exemple monodimensionnel de la figure 18.1. En ce qui concerne le réglage de la valeur de α , on peut par exemple utiliser la technique de la descente maximale, qui calcule au mieux cette valeur, ou en prendre une approximation (en général, faire décroître α avec n). On peut également faire intervenir la dérivée seconde de f .

Il est plus difficile d'éviter les minima locaux. On peut évidemment appliquer plusieurs fois la méthode en faisant varier les paramètres de l'algorithme (réglage de α , point de départ) et choisir le minimum des minima locaux trouvés. On peut aussi introduire des perturbations aléatoires dans le calcul de la suite u_n , pour tenter de la faire « sortir » des zones où elle converge vers un minimum local.

[Fle80] est une bonne référence pour la théorie et la pratique des méthodes d'optimisation (en particulier par descente de gradient).

18.3 La rétropropagation du gradient de l'erreur

Dans cette annexe, nous montrons comment établir les formules de la section 10.3.2 qui dictent le changement de poids d'un réseau de neurones entraîné par la règle de la rétropropagation de l'erreur.

18.3.1 Notations

Nous notons σ_j l'état d'activation du neurone j à la présentation du vecteur d'entrée \mathbf{x} . De la même façon, nous notons y_j la sortie du neurone j , $w(i, j)$ le poids de la connexion entre les neurones i et j , et u_j la valeur désirée du neurone de sortie j .

De plus, nous notons $source(j)$ l'ensemble des neurones connectés au neurone j , $dest(j)$ l'ensemble des neurones auxquels j se connecte, et $sortie(j)$ l'ensemble des neurones de sortie.

18.3.2 Fonctionnement du système

Rappelons tout d'abord les équations de base régissant le fonctionnement du réseau. La règle de propagation du réseau est la suivante :

$$\sigma_j = \sum_{i \in source(j)} w(i, j) \cdot y_i \quad (18.15)$$

La fonction de sortie permet de calculer y_j en fonction de σ_j :

$$y_j = f(\sigma_j) \quad (18.16)$$

Cette fonction doit être continue, croissante, bornée et dérivable. On utilise souvent une *fonction sigmoïde* définie par :

$$f(u) = \frac{1}{1 + e^{-\lambda u}} \quad (18.17)$$

avec λ réel positif.

Cette fonction répond à l'équation différentielle :

$$f' = \lambda f(1 - f)$$

En effet :

$$f(u) = \frac{1}{1 + e^{-\lambda u}}$$

$$f'(u) = \frac{-(-\lambda e^{-\lambda u})}{(1 + e^{-\lambda u})^2}$$

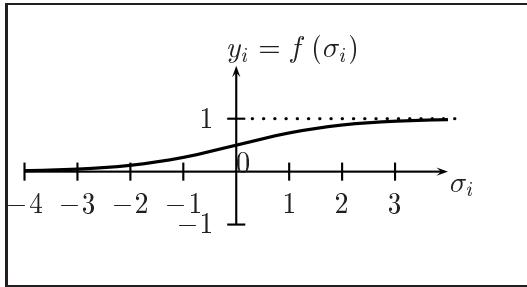


FIG. 18.2 – La fonction sigmoïde d'équation $y_i = f(\sigma_i) = \frac{1}{1+e^{-\lambda\sigma_i}}$. Cette fonction est paramétrée par sa pente à l'origine λ . Pour λ très grand, on retrouve la fonction seuil. Pour λ très petit, cette fonction est pratiquement linéaire dans une vaste région autour de l'origine. La sigmoïde de cette figure est dessinée pour $\lambda = 1$.

$$f'(u) = \lambda \frac{1}{1+e^{-\lambda u}} \left(1 - \frac{1}{1+e^{-\lambda u}}\right) = \lambda f(u)(1 - f(u)) \quad (18.18)$$

On peut remarquer que, quand λ est très grand, f se rapproche de la fonction signe ; quand λ est très petit, mais non nul, f est pratiquement linéaire autour de l'origine et de valeurs bornées entre 0 et 1.

On considérera ici pour simplifier la fonction sigmoïde de paramètre λ égal à 1. Prendre λ quelconque introduirait simplement un coefficient de proportionnalité dans les calculs qui suivent.

18.3.3 Calcul du gradient

Nous cherchons à faire évoluer le réseau de façon à minimiser un coût $E_{\mathbf{x}}$ pour chaque vecteur d'entrée \mathbf{x} . On utilise ici le critère des moindres carrés :

$$E_{\mathbf{x}} = \frac{1}{2} \sum_{j \in \text{sortie}} (u_j - y_j)^2 \quad (18.19)$$

Nous cherchons le gradient de $E_{\mathbf{x}}$ par rapport aux poids du réseau, c'est-à-dire les valeurs

$$\frac{\partial E_{\mathbf{x}}}{w(i, j)}$$

Pour plus de clarté, nous pouvons omettre l'indice \mathbf{x} dans les équations qui viennent.

Par la règle de dérivation chaînée, on sait que :

$$\frac{\partial E}{\partial w(i, j)} = \frac{\partial E}{\partial \sigma_j} \cdot \frac{\partial \sigma_j}{\partial w(i, j)}$$

De l'équation (18.15), on déduit que :

$$\frac{\partial \sigma_j}{\partial w(i, j)} = y_i$$

Par conséquent, en notant :

$$\delta_j = \frac{\partial E}{\partial \sigma_j}$$

on a :

$$\frac{\partial E}{\partial w(i, j)} = y_i \cdot \delta_j$$

Le cas des neurones de sortie

Lorsque j est un neurone de sortie, avec f définie comme dans l'équation (18.17) et λ égal à 1, on a :

$$\delta_j = \frac{\partial E}{\partial \sigma_j} = \frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial w(i, j)}$$

On a, d'après les équations 18.16 et 18.18 :

$$\frac{\partial y_j}{\partial \sigma_j} = y_j \cdot (1 - y_j)$$

Puisque j est un neurone de sortie, d'après l'équation 18.19 :

$$\frac{\partial E}{\partial y_j} = y_j - u_j$$

Donc, pour un neurone de sortie :

$$\delta_j = y_j \cdot (1 - y_j) \cdot (y_j - u_j)$$

Le cas des neurones cachés

Lorsque j est un neurone caché, on calcule δ_j par *rétrôpropagation*. La remarque fondamentale est que la contribution de chaque neurone formel j sur la sortie est propagée vers la sortie à travers les éléments de la couche $dest(j)$. Par conséquent :

$$\delta_j = \frac{\partial E}{\partial \sigma_j} = \sum_{k \in dest(j)} \frac{\partial E}{\partial \sigma_k} \cdot \frac{\partial \sigma_k}{\partial \sigma_j}$$

En développant les calculs :

$$\begin{aligned} \delta_j &= \sum_{k \in dest(j)} \delta_k \cdot \frac{\partial \sigma_k}{\partial \sigma_j} \\ &= \sum_{k \in dest(j)} \delta_k \cdot \frac{\partial \sigma_k}{\partial y_j} \cdot \frac{\partial y_j}{\partial \sigma_j} \\ &= \sum_{k \in dest(j)} \delta_k \cdot w(j, k) \cdot y_j \cdot (1 - y_j) \\ &= y_j \cdot (1 - y_j) \cdot \sum_{k \in dest(j)} \delta_k \cdot w(j, k) \end{aligned}$$

Conclusion

En combinant les équations précédentes, on trouve les valeurs :

- Pour les neurones de sortie :

$$\frac{\partial E}{\partial w(i, j)} = y_i \cdot \delta_j = y_i \cdot y_j \cdot (1 - y_j) \cdot (y_j - u_j)$$

- Pour les neurones cachés :

$$\frac{\partial E}{\partial w(i,j)} = y_i \cdot \delta_j = y_i \cdot y_j \cdot (1 - y_j) \cdot \sum_{k \in \text{dest}(j)} \delta_k \cdot w(j, k)$$

Finalement, pour modifier $w(i, j)$, il faut lui additionner une quantité dans la direction opposée au gradient :

$$\Delta w(i, j) = -\alpha \frac{\partial E}{\partial w(i, j)} = -\alpha \cdot y_i \cdot \delta_j$$

où α , compris entre 0 et 1 est le pas de déplacement, aussi appelé le taux d'apprentissage.

18.4 Estimation d'une densité de probabilité en un point

Soit une distribution de probabilités de densité p dans \mathbb{R}^d . La probabilité P qu'un vecteur x tiré selon cette distribution se trouve à l'intérieur d'une région fermée \mathcal{R} vaut :

$$P = \int_{\mathcal{R}} p(y) dy \quad (18.20)$$

Soient $\{x_1, x_2, \dots, x_m\}$ m points tirés indépendamment selon p . La probabilité que k d'entre eux se trouvent à l'intérieur de la région est donnée par :

$$P(k) = C_m^k P^k (1 - P)^{m-k}$$

et l'espérance mathématique de cette variable $P(k)$ vaut :

$$E(P(k)) = mP$$

Par conséquent, $E(P(k))$ est une estimation de la probabilité de l'événement : « parmi les m points, k sont dans la région \mathcal{R} ». On peut montrer que c'est un bon estimateur, dans la mesure où sa variance diminue rapidement avec l'augmentation de m . On peut donc écrire, pour m assez grand :

$$P \approx k/N \quad (18.21)$$

En supposant p continue dans la région \mathcal{R} et celle-ci assez petite pour que p n'y varie presque pas, on a d'autre part, en notant V le volume de la région \mathcal{R} :

$$\int_{\mathcal{R}} p(y) dy \approx p(x)V \quad (18.22)$$

Les équations précédentes mènent à :

$$p(x) \approx \frac{P}{V} \approx \frac{k/m}{V}$$

En pratique, pour assurer la qualité des approximations, on fait dépendre la région \mathcal{R} du nombre de points m : on la note maintenant \mathcal{R}_m et son volume V_m ; elle inclut k_m points parmi les m . On a donc l'estimation suivante :

$$p(x) \approx P_m(x) = \frac{k_m/m}{V_m}$$

Pour que $P_m(x)$ converge vers $p(x)$, il faut :

- assurer que l'approximation de l'équation 18.22 soit valide, donc avoir :

$$\lim_{m \rightarrow \infty} V_m = 0$$

- assurer que celle 18.21 le soit aussi, donc avoir :

$$\lim_{m \rightarrow \infty} k_m = \infty$$

- enfin, pour que $p(x)$ reste à une valeur finie :

$$\lim_{m \rightarrow \infty} \frac{k_m}{m} = 0$$

18.5 L'estimation des paramètres d'une distribution gaussienne

On se place ici, pour éviter des formules fastidieuses, dans un espace monodimensionnel. Une distribution gaussienne dépend alors seulement de deux paramètres scalaires : sa moyenne μ et sa variance σ . La variable x est aussi un scalaire, c'est pourquoi elle n'est pas notée ici comme un vecteur \boldsymbol{x} . Elle s'écrit :

$$\mathcal{N}(x, \mu, \sigma) = \left(\frac{1}{\sqrt{2\pi}\sigma}\right) \exp\left((-1/2)\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

ou :

$$\text{Log}(\mathcal{N}(x, \mu, \sigma)) = -\text{Log}(\sigma) - \text{Log}(2\pi) - \frac{(x-\mu)^2}{2\sigma^2}$$

L'estimateur au maximum de vraisemblance suppose que tous les échantillons d'apprentissage $\mathcal{S} = \{x_1, x_2, \dots, x_m\}$ sont tirés indépendamment. On cherche donc dans l'ensemble des distributions gaussiennes de paramètres σ et μ celle qui maximise le produit de ses valeurs sur les points d'apprentissage.

Ceci est équivalent à trouver les paramètres pour lesquels l'expression :

$$T(\mu, \sigma, m) = \sum_{k=1}^m \text{Log}(\mathcal{N}(x_k, \mu, \sigma))$$

est maximale.

On est donc amené à chercher les valeurs μ et σ telles que :

$$\frac{\partial}{\partial \mu} T(\mu, \sigma, m) = 0$$

$$\frac{\partial}{\partial \sigma} T(\mu, \sigma, m) = 0$$

d'où :

$$\sum_{k=1}^m \frac{x_k - \mu}{\sigma^2} = 0$$

$$-m/\sigma + \sum_{k=1}^m \frac{x_k - \mu}{\sigma^2} = 0$$

Finalement, la solution au problème de l'estimation au maximum de vraisemblance est donnée par les valeurs suivantes :

$$\hat{\mu} = \frac{1}{m} \sum_{k=1}^m x_k$$

et :

$$\hat{\sigma} = \sqrt{\frac{1}{m} \sum_{k=1}^m (x_k - \hat{\mu})^2}$$

En revenant au cas général multiclassé en dimension d , il faut estimer au maximum de vraisemblance pour chaque classe de numéro i un *vecteur moyenne* $\boldsymbol{\mu}_i$ et une *matrice de covariance* Q_i ; on retrouverait, en développant des calculs analogues, les formules :

$$\widehat{\boldsymbol{\mu}}_i = \frac{1}{m_i} \sum_{k=1}^{m_i} x_k$$

et :

$$\widehat{Q}_i = \frac{1}{m_i} \sum_{k=1}^{m_i} (x_k - \widehat{\boldsymbol{\mu}}_i)(x_k - \widehat{\boldsymbol{\mu}}_i)^T$$

18.6 Pourquoi et comment la règle du PPV converge-t-elle ?

18.6.1 Pourquoi ?

Soit $\mathcal{S} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m\}$ l'échantillon d'apprentissage, \mathbf{x} le point dont on cherche la classe par la règle du $1 - ppv$ et \mathbf{x}_0 le point de X le plus proche de \mathbf{x} . Notons $B(\mathbf{x}, \rho)$ la sphère de rayon ρ centrée en \mathbf{x} .

La probabilité qu'un point de \mathcal{S} se trouve dans $B(\mathbf{x}, \rho)$ vaut :

$$P(\rho) = \int_{B(\mathbf{x}, \rho)} p(\mathbf{x}) d\mathbf{x} \quad (18.23)$$

La probabilité qu'aucun point de l'ensemble d'apprentissage \mathcal{S} ne se trouve dans $B(\mathbf{x}, \rho)$ est égale à $(1 - P(\rho))^m$ qui tend vers zéro quand m augmente. Par conséquent, \mathbf{x}_0 tend en probabilité vers \mathbf{x} , ce qui assure la convergence désirée.

18.6.2 Comment ?

L'erreur moyenne réalisée par la règle du $1 - ppv$ peut se calculer en remarquant qu'en attribuant à \mathbf{x} la classe de \mathbf{x}_0 , on commet l'erreur :

$$err_{ppv} = P[\omega(\mathbf{x}) \neq \omega(\mathbf{x}_0)] = \sum_{i=1}^d P[\mathbf{x} \in \omega_i, \mathbf{x}_0 \notin \omega_i] = \sum_{i=1}^d P(\omega_i | \mathbf{x})[1 - P(\omega_i | \mathbf{x}_0)] \quad (18.24)$$

Quand la taille m de l'ensemble d'apprentissage augmente, on a vu ci-dessus que \mathbf{x}_0 tend en probabilité vers \mathbf{x} , ce qui implique que $P(\omega_i | \mathbf{x}_0)$ tend vers $P(\omega_i | \mathbf{x})$. Par conséquent :

$$\lim_{m \rightarrow \infty} err_{ppv} = \int_{\mathbb{R}^d} \sum_{i=1}^d P(\omega_i | \mathbf{x})[1 - P(\omega_i | \mathbf{x})]p(\mathbf{x})d\mathbf{x} \quad (18.25)$$

Prenons le cas à deux classes ($C = 2$). Soit $\omega_1(\mathbf{x})$ la classe que donnerait la décision bayésienne et $\omega_2(\mathbf{x})$ l'autre. L'erreur bayésienne vaut :

$$err^* = err(\omega_1) = \int_{\mathbb{R}^d} p(\omega_2(\mathbf{x}) | \mathbf{x})p(\mathbf{x})d\mathbf{x} \quad (18.26)$$

L'erreur par la décision du $1 - ppv$, en vertu du calcul précédent, vaut pour $C = 2$:

$$err_{ppv} = P(\omega_1 | \mathbf{x})[1 - P(\omega_1 | \mathbf{x})] + P(\omega_2 | \mathbf{x})[1 - P(\omega_2 | \mathbf{x})] \quad (18.27)$$

d'où :

$$err_{ppv} = 2P(\omega_2(\mathbf{x}) | \mathbf{x})[1 - P(\omega_2(\mathbf{x}) | \mathbf{x})] \leq 2P(\omega_2(\mathbf{x}) | \mathbf{x}) \quad (18.28)$$

D'où la formule dans le cas de deux classes :

$$\lim_{m \rightarrow \infty} err_{ppv} \leq 2err^* \quad (18.29)$$

18.7 Le calcul de l'intervalle de confiance de l'estimation de la probabilité d'une règle de classification

Étant donnés un taux d'erreur réel $R_{Réel}(h)$, et un échantillon de test de taille t , le nombre d'erreurs t_{err} mesuré sur un échantillon \mathcal{T} distribué de manière i.i.d. (suivant la même distribution que l'échantillon d'apprentissage \mathcal{S}) suit une loi binomiale :

$$\begin{aligned} P(t_{err} | R_{Réel}(h), t) &= \mathcal{B}(t_{err} | R_{Réel}(h), t) \\ &\triangleq \frac{t_{err}! (t - t_{err})!}{t!} R_{Réel}(h)^{t_{err}} (1 - R_{Réel}(h))^{(t - t_{err})} \end{aligned}$$

L'expression ci-dessus donne la probabilité que t_{err} observations dans un échantillon de test de taille t soient mal classées, étant donné que le taux d'erreur réel est $R_{Réel}(h)$. En utilisant le théorème de Bayes, nous pouvons « retourner » cette équation pour obtenir ce qui nous intéresse, c'est-à-dire une estimation du taux d'erreur réel connaissant le nombre d'erreurs mesuré sur \mathcal{T} :

$$P(R_{Réel}(h) | t_{err}, t) = \frac{P(t_{err} | R_{Réel}(h), t)}{\int P(t_{err} | R_{Réel}(h), t)}$$

En supposant que $P(R_{Réel}(h), t)$ ne varie pas avec $R_{Réel}(h)$ et que $P(t_{err} | R_{Réel}(h), t)$ suit une distribution binomiale, nous avons une distribution bêta pour $R_{Réel}(h)$:

$$\begin{aligned} P(R_{Réel}(h) | t_{err}, t) &= B_e(R_{Réel}(h)) | t_{err} + 1, t - t_{err} + 1 \\ &\triangleq \frac{R_{Réel}(h)^{t_{err}} (1 - R_{Réel}(h))^{(t - t_{err})}}{\int R_{Réel}(h)^{t_{err}} (1 - R_{Réel}(h))^{(t - t_{err})} d_{R_{Réel}(h)}} \end{aligned}$$

où $B_e(x|\alpha, \beta) = [\Gamma(\alpha+\beta)/(\Gamma(\alpha)\Gamma(\beta))] x^{\alpha-1} (1-x)^{\beta-1}$. Cette densité de probabilité *a posteriori* fournit toute l'information qui peut être tirée de l'erreur mesurée $\hat{R}_{Réel}(h)$. On utilise cependant généralement un résumé de cette information sous la forme d'intervalles de confiance autour d'une valeur estimée.

Si les échantillons d'apprentissage et de test aléatoires sont indépendants alors la précision de l'estimation ne dépend que du nombre t d'exemples de l'ensemble de test et de la valeur de $\hat{R}_{Réel}(h)$. Une approximation suffisante dans le cas où t est assez grand (au delà de la centaine) donne l'*intervalle de confiance* de $\hat{R}_{Réel}(h)$ à $x\%$ par la formule:

$$\left[\frac{t_{err}}{t} \pm \zeta(x) \sqrt{\frac{\frac{t_{err}}{t} (1 - \frac{t_{err}}{t})}{t}} \right]$$

La fonction $\zeta(x)$ a en particulier les valeurs suivantes:

x	50 %	68 %	80 %	90 %	95 %	98 %	99 %
$\zeta(x)$	0.67	1.00	1.28	1.64	1.96	2.33	2.58

18.8 Pourquoi la règle de décision bayésienne est-elle optimale?

Appelons h^* la règle de décision bayésienne, celle qui affecte au point \mathbf{x} la classe:

$$\omega^*(\mathbf{x}) = \text{ArgMax}(P(\omega_i | \mathbf{x}))$$

Soit h_0 la règle idéale de décision, affectant \mathbf{x} à sa vraie classe $h_0(\mathbf{x})$ et soit h une règle de décision quelconque affectant \mathbf{x} à la classe $h(\mathbf{x})$. On cherche à montrer que l'erreur moyenne $err(h^*)$ commise par h^* est inférieure ou égale à $err(h)$, commise par h , soit:

$$\int_{\mathbb{R}^d} P(h^*(\mathbf{x}) \neq h_0(\mathbf{x})) p(\mathbf{x}) d\mathbf{x} \leq \int_{\mathbb{R}^d} P(h(\mathbf{x}) \neq h_0(\mathbf{x})) p(\mathbf{x}) d\mathbf{x} \quad (18.30)$$

Or, pour toute règle h :

$$P(h(\mathbf{x}) \neq h_0(\mathbf{x})) = 1 - P(h(\mathbf{x}) | \mathbf{x}) \quad (18.31)$$

donc :

$$err(h) = 1 - \int_{\mathbb{R}^d} P(h(\mathbf{x}) | \mathbf{x}) p(\mathbf{x}) d\mathbf{x} \quad (18.32)$$

De la définition de la règle h^* , on déduit immédiatement:

$$\forall \mathbf{x} \in \mathbb{R}^d, P(h^*(\mathbf{x}) | \mathbf{x}) \geq P(h(\mathbf{x}) | \mathbf{x}) \quad (18.33)$$

Par conséquent, la valeur:

$$err(h) - err(h^*) = 1 - \int_{\mathbb{R}^d} (P(h(\mathbf{x}) | \mathbf{x}) - P(h^*(\mathbf{x}) | \mathbf{x})) p(\mathbf{x}) d\mathbf{x} \quad (18.34)$$

est toujours positive ou nulle, ce qui établit la relation cherchée.

18.9 Apprentissage par estimation-maximisation.

L'algorithme *estimation-maximisation* (*EM*) est une procédure générale pour apprendre la valeur de paramètres cachés de certains processus probabilistes. C'est en particulier en l'utilisant que les paramètres des HMM peuvent être appris (voir le chapitre 13). Mais cette procédure est aussi utile dans l'apprentissage des réseaux bayésiens (chapitre 12) et dans la classification non supervisée (chapitre 15). Nous allons le présenter ici sur un exemple simple, puis revenir sur son application à l'apprentissage des HMM et au mélange de distributions gaussiennes.

18.9.1 Un exemple

Nous disposons de deux pièces de monnaie truquées A et B , avec des probabilités respectives p_A et p_B de tomber sur *Pile* et $1 - p_A$ et $1 - p_B$ de tomber sur *Face*. Nous connaissons les valeurs p_A et p_B . Nous demandons à un huissier de procéder en secret aux opérations suivantes :

- tirer un nombre μ au hasard entre 0 et 1;
- répéter N fois les manipulations suivantes :
 - choisir la pièce A avec la probabilité μ ou la pièce B avec la probabilité $(1 - \mu)$,
 - lancer la pièce choisie,
 - enregistrer le résultat du lancer : *Pile* ou *Face*.

Une fois l'affaire terminée, l'huissier nous communique la séquence des N valeurs *Pile* ou *Face* qu'il a notées. Notre problème est alors le suivant : estimer la valeur du paramètre caché μ à partir de la suite $\mathcal{O} = (O_1, \dots, O_N)$ de ces observations. Chaque observation O_i vaut donc *Pile* ou *Face*.

Si nous savions quelle pièce a été utilisée à chaque lancer, nous pourrions estimer μ par la valeur :

$$\frac{N_A}{N} = \frac{\text{Nombre de fois que la pièce } A \text{ a été lancée}}{N}$$

Mais ce n'est pas le cas. Nous ne connaissons que le résultat des tirages. Nous pouvons cependant calculer μ par une technique itérative, *l'algorithme EM*².

18.9.2 Application de l'algorithme *EM* à l'exemple

Initialisation

Donner une valeur arbitraire μ_0 strictement comprise entre 0 et 1.

Étape p

- Estimation

On utilise les observations \mathcal{O} pour calculer une estimation de N_A .

On note :

- μ_p l'estimation courante de μ ,
- $P_A(O_i)$ la probabilité de l'événement : « A est sortie au tirage i »,
- $P_B(O_i)$ la probabilité de l'événement : « B est sortie au tirage i »,
- $E_p(A | \mathcal{O})$ l'estimation du nombre N_A de tirages de la pièce A .

Cette dernière valeur peut se calculer en fonction des trois précédentes par l'expression :

$$E_p(A | \mathcal{O}) = \sum_{i=1}^{i=N} \frac{\mu_p P_A(O_i)}{\mu_p P_A(O_i) + (1 - \mu_p) P_B(O_i)}$$

2. *E* pour Estimation (ou en anglais *Expectation*), *M* pour Maximisation.

Mais les valeurs $P_A(O_i)$ et $P_B(O_i)$ sont inconnues. C'est le centre de l'algorithme que de savoir les estimer.

- Maximisation

On calcule une nouvelle estimation de μ :

$$\mu_{p+1} = \frac{E_p(A | \mathcal{O})}{N}$$

Test d'arrêt

$$\mu_{p+1} \approx \mu_p$$

Pour résoudre le problème, il faut donc déterminer les valeurs $P_A(O_i)$ et $P_B(O_i)$. On utilise pour cela la notion de *statistique suffisante*.

18.9.3 Statistique suffisante

Pour notre exemple, l'ordre dans lesquels les O_i sont apparus dans l'ensemble des observations \mathcal{O} n'a pas d'importance: le nombre P d'observations *Pile* dans \mathcal{O} suffit. On dit qu'il s'agit d'une *statistique suffisante* pour estimer le paramètre caché μ . Cette observation permet de calculer explicitement $E_p(\mathcal{A} | \mathcal{O})$ de la façon suivante.

On décompose d'abord la formule précédente sur les tirages ayant donné *Pile* et ceux ayant donné *Face*, en nombres respectifs P et F .

$$\begin{aligned} E_p(A | \mathcal{O}) &= \sum_{i=1}^P \frac{\mu_p P_A(Pile)}{\mu_p P_A(Pile) + (1 - \mu_p) P_B(Pile)} \\ &\quad + \sum_{i=1}^F \frac{\mu_p P_A(Face)}{\mu_p P_A(Face) + (1 - \mu_p) P_B(Face)} \end{aligned}$$

En utilisant maintenant le fait que les tirages sont indépendants:

$$\begin{aligned} E_p(A | \mathcal{O}) &= \sum_{i=1}^P \frac{\mu_p p_A}{\mu_p p_A + (1 - \mu_p) p_B} + \sum_{i=1}^F \frac{\mu_p p_A}{\mu_p p_A + (1 - \mu_p) p_B} \\ &= P \frac{\mu_p p_A}{\mu_p p_A + (1 - \mu_p) p_B} + F \frac{\mu_p p_A}{\mu_p p_A + (1 - \mu_p) p_B} \end{aligned}$$

On a donc maintenant une formule pour calculer $E_p(A | \mathcal{O})$ en fonction de μ_p et de valeurs connues.

18.9.4 Plus généralement

On démontre que l'algorithme *EM* décrit en 18.1 fait croître la vraisemblance de Λ_p vis-à-vis des données. Par conséquent, il converge vers un optimum local.

18.9.5 Retour sur l'exemple

Prenons $p_A = 0.2$ et $p_B = 0.6$. L'huissier effectue $N = 100$ tirages et fournit une série \mathcal{O} d'observations comportant $P = 50$ *Pile* et $F = 50$ *Face*. Quelle est la valeur de μ que produit l'algorithme *EM*?

Algorithme 18.1 Estimation-maximisation

Définir une statistique suffisante pour estimer l'ensemble Λ des paramètres cachés

Initialiser Λ

$p \leftarrow 1$

tant que $\Lambda_p \neq \Lambda_{p+1}$ **faire**

ESTIMATION

Utiliser les observations pour calculer la statistique suffisante de Λ_p

MAXIMISATION

Calculer Λ_{p+1} comme une estimation au maximum de vraisemblance de Λ à partir des résultats de l'étape p d'estimation

$p \leftarrow p + 1$

fin tant que

La récurrence de base est la suivante, en regroupant les deux phases en une seule :

$$\mu_{p+1} = \frac{1}{P+F} \left(P \frac{\mu_p p_A}{\mu_p p_A + (1 - \mu_p) p_B} + F \frac{\mu_p (1 - p_A)}{\mu_p (1 - p_A) + (1 - \mu_p) (1 - p_B)} \right)$$

D'où, pour deux initialisations de μ :

μ_0	0.800	0.100
μ_1	0.730	0.109
μ_2	0.659	0.118
μ_3	0.593	0.127
μ_4	0.536	0.135
μ_5	0.488	0.144
μ_{10}	0.351	0.183
μ_{20}	0.273	0.228
μ_{30}	0.256	0.243
μ_{40}	0.252	0.248
μ_{50}	0.250	0.249
μ_{60}	0.250	0.250

Ce résultat est conforme à l'intuition : il faut que l'huissier ait tiré en moyenne quatre fois plus souvent la pièce B que la pièce A pour avoir rétabli l'équilibre entre le nombre de *Pile* et celui de *Face*.

Il se trouve que dans cet exemple très simple, on peut en réalité estimer directement μ car la proportion de *Pile* est en effet une estimation de :

$$\mu p_A + (1 - \mu) p_B$$

Dans notre application numérique :

$$1/2 = \mu 0.2 + (1 - \mu) 0.6$$

d'où :

$$\mu = 0.25$$

18.9.6 L'apprentissage des paramètres des HMM

Les formules de réestimation des paramètres de modèles de Markov cachés donnés au chapitre 13, bien que beaucoup plus compliquées en apparence, ne sont pas fondamentalement différentes de celles présentées ci-dessus.

Prenons le cas de la réestimation d'une probabilité de transition, donnée par la formule³ :

$$\bar{a}_{ij} = \frac{\sum_{k=1}^N \sum_{t=1}^{|O^k|-1} \xi_t^k(i, j)}{\sum_{k=1}^N \sum_{t=1}^{|O^k|-1} \gamma_t^k(i)}$$

avec :

- $\xi_t^k(i, j)$ la probabilité, étant donnés une phrase O^k et un HMM Λ , que ce soit l'état s_i qui ait émis la lettre de rang t de O^k et l'état s_j qui ait émis celle de rang $t+1$
- $\gamma_t^k(i)$ la probabilité que la lettre de rang t de la phrase O^k ait été émise par l'état s_j .

\bar{a}_{ij} , la valeur réestimée de a_{ij} , est une composante du paramètre caché $\Lambda = (A, B, \pi)$ qui définit le HMM cherché. Elle est calculée par un comptage sur la base d'apprentissage \mathcal{O} ; ce comptage calcule une statistique suffisante pour estimer la probabilité qu'est réellement a_{ij} . En effet, \bar{a}_{ij} compte la proportion des transitions empruntées entre s_i et s_j parmi toutes celles qui quittent s_i , dans l'émission de chaque phrase O^k de \mathcal{O} .

La normalisation assure que sur leur ensemble, les \bar{a}_{ij} somment à 1.

De la sorte, l'algorithme *EM* réestime tous les paramètres cachés du HMM cherché. Comme les formules de réestimation sont conformes aux hypothèses, la convergence vers un optimum (éventuellement local) est assurée.

18.9.7 L'apprentissage des paramètres de distributions multigaussiennes

On dispose d'une collection $\mathcal{S} = \{\mathbf{x}_1, \dots, \mathbf{x}_m\}$ d'exemples, qui sont des vecteurs de \mathbb{R}^d . On fait l'hypothèse que ces vecteurs sont des tirages aléatoires d'un *mélange* de k distributions gaussiennes $\mathcal{N}_1, \dots, \mathcal{N}_k$. On cherche à estimer les paramètres de chaque distribution, ainsi que la façon dont elles sont mélangées.

Le tirage d'un exemple pourrait se décrire ainsi : d'abord, choisir aléatoirement une des k distributions. Ensuite, tirer l'exemple selon cette distribution. Par conséquent, pour définir le mélange, il suffit de se donner les k valeurs qui sont les probabilités que l'une des k distributions soit tirée.

Un exemple \mathbf{x}_i doit donc se décrire de manière plus complète par :

$$y_i = (\mathbf{x}_i, \mathbf{z}_{i1}, \dots, \mathbf{z}_{ik})$$

où :

- \mathbf{x}_i est le vecteur observé,
- pour $j = 1, k$, \mathbf{z}_{ij} vaut 1 si \mathbf{x}_i a été généré par \mathcal{N}_i . Les valeurs \mathbf{z}_{ij} sont cachées à l'observateur.

3. On se reporterà au chapitre 13 pour le détail des notations.

Pour simplifier, nous prenons $d = 1$. Nous supposons aussi que les k distributions, de moyennes μ_1, \dots, μ_k , ont la même variance σ . La méthode se généralise sans trop de difficultés à d quelconque et à des matrices de covariances différentes pour chaque gaussienne. L'algorithme *EM* s'applique comme décrit en 18.2. Il a pour résultat le vecteur $h = (\mu_1, \dots, \mu_k)$ et les estimations des valeurs z_{i1}, z_{ik} . Ces dernières quantités sont donc les probabilités avec lesquelles on tire les gaussiennes \mathcal{N}_j , pour $j = 1, k$.

Algorithme 18.2 Mélange de k gaussiennes

Initialiser aléatoirement $h = (\mu_1, \dots, \mu_k)$

tant que le processus n'a pas convergé **faire**

Estimation

Calculer les estimations $E(z_{i1})$ de $z_{i1}, \dots E(z_{ik})$ de z_{ik} en supposant que l'hypothèse courante sur h est la bonne:

pour $j = 1, k$ **faire**

$$E(z_{ij}) \leftarrow \frac{p(x = \mathbf{x}_i \mid \mu = \mu_j)}{\sum_{n=1}^k p(x = \mathbf{x}_i \mid \mu = \mu_n)}$$

Donc :

$$E(z_{ij}) \leftarrow \frac{e^{-\frac{1}{2\sigma^2}(\mathbf{x}_i - \mu_j)^2}}{\sum_{n=1}^k e^{-\frac{1}{2\sigma^2}(\mathbf{x}_i - \mu_n)^2}}$$

fin pour

Maximisation

Calculer une nouvelle estimation de $h = (\mu_1, \dots, \mu_k)$ en supposant que z_{ij} est égale à $E(z_{ij})$

pour $j = 1, k$ **faire**

$$\mu_j \leftarrow \frac{\sum_{i=1}^N E(z_{ij}) \mathbf{x}_i}{\sum_{i=1}^N E(z_{ij})}$$

fin pour

fin tant que

Bibliographie générale

- [AB92] M. Anthony and N. Biggs. *Computational Learning Theory*. Cambridge University Press, 1992.
- [BB82] D. Ballard and C. Brown. *Computer Vision*. Prentice-Hall, 1982.
- [BB92] A. Belaïd and Y. Belaïd. *Reconnaissance des formes*. InterEditions, 1992.
- [BFOS84] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth Inc., 1984.
- [Bis95] C. Bishop. *Neural networks for pattern recognition*. Oxford University Press, 1995.
- [BN99] A. Becker and P. Naïm. *Les réseaux bayésiens : modèles graphiques de la connaissance*. Eyrolles, 1999.
- [BNKF98] W. Banzhaf, P. Nordin, R. Keller, and F. Francone. *Genetic Programming: An introduction*. Morgan Kaufmann, 1998.
- [CL96] G. Caraux and Y. Lechevallier. Règles de décision de Bayes et méthodes statistiques de discrimination. *Revue d'Intelligence Artificielle*, 10(2-3):219–283, 1996.
- [CM98] V. Cherkassky and F. Mulier. *Learning from data. Concepts, theory and methods*. Wiley Interscience, 1998.
- [CST00] N. Cristianini and J. Shawe-Taylor. *Support vector machines and other kernel-based learning methods*. Cambridge University Press, 2000.
- [CT91] T. Cover and A. Thomas. *Elements of Information Theory*. Wiley, 1991.
- [DH73] R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. Wiley, 1973.
- [DHS01] R. Duda, P. Hart, and D. Stork. *Pattern classification*. Wiley, 2001.
- [DKBM00] E. Diday, Y. Kodratoff, P. Brito, and M. Moulet, editors. *Induction symbolique numérique à partir de données*. Cepadues, 2000.
- [DMS⁺02] G. Dreyfus, J. Martinez, M. Samuelides, M. Gordon, F. Badran, S. Thiria, and L. Héault. *Réseaux de neurones*. Eyrolles, 2002.
- [Dup94] J-P. Dupuy. *Aux origines des sciences cognitives*. Éditions de la Découverte, 1994.
- [EdB01] A. Engel and C. Van den Broeck. *Statistical mechanics of learning*. Cambridge University Press, 2001.
- [Fle80] R. Fletcher. *Practical methods of optimization*. Wiley, 1980.
- [Fog98] L. Fogel, editor. *Evolutionary computation: towards a new philosophy of machine intelligence*. IEEE Press, 1998.
- [Fre98] B. Frey. *Graphical Models for Machine Learning and Digital Communications*. Bradford Book, The MIT Press, 1998.
- [Gol89] D. Goldberg. *Genetic Algorithm in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.
- [Hay99] S. Haykin. *Neural networks : A comprehensive foundation*. Prentice Hall, 1999.
- [Hou91] J. Houston. *Fundamentals of Learning and Memory*. Harcourt Brace Jovanovich, 1991. Fourth Edition.
- [HTF02] S. Hastie, T. Tibshirani, and J. Friedman. *The elements of statistical learning*. Springer Verlag, 2002.
- [JD88] A. Jain and R. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, 1988.

-
- [Kod86] Y. Kodratoff. *Lecons d'apprentissage symbolique automatique*. Cepaduès, 1986.
- [KV94] M. Kearns and U. Vazirani. *An introduction to computational learning theory*. MIT Press, 1994.
- [Lan96] P. Langley. *Elements of Machine Learning*. Morgan Kaufmann, 1996.
- [MCM83] R. Michalski, J. Carbonnel, and T. Mitchell. *Machine Learning, An Artificial Intelligence Approach*, volume 1. Tioga, 1983.
- [MCM86] R. Michalski, J. Carbonnel, and T. Mitchell. *Machine Learning, An Artificial Intelligence Approach*, volume 2. Morgan-Kaufmann, 1986.
- [Mic84] L. Miclet. *Méthodes structurelles pour la reconnaissance des formes*. Eyrolles, 1984.
- [Mil93] M. Milgram. *Reconnaissance des formes*. Armand Colin, 1993.
- [Mit96] M. Mitchell. *An introduction to genetic algorithms*. MIT Press, 1996.
- [Mit97] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.
- [MK89] R. Michalski and Y. Kodratoff. *Machine Learning, An Artificial Intelligence Approach*, volume 3. Morgan-Kaufmann, 1989.
- [Nic93] J. Nicolas, editor. *Support de cours de l'Ecole CNRS sur l'Apprentissage Automatique*, 1993. St Raphaël, 26-27 Mars 1993. Contributions de: O. Gascuel, L. Miclet, J. Nicolas, J. Quinton, C. Rouveiro et C. Vrain. Disponible à l'IRISA auprès de J. Nicolas.
- [Nil98] N. Nilsson. *Artificial Intelligence : a new synthesis*. Morgan-Kaufmann, 1998.
- [NS93] A. Nerode and R. Shore. *Logic for Applications*. Springer-Verlag, 1993.
- [Pea88] J. Pearl. *Probabilistic reasoning in intelligent systems*. Morgan-Kaufmann, 1988.
- [Pea00] J. Pearl. *Causality : Models, Reasoning and Inference*. Cambridge University Press, 2000.
- [RN95] S. Russel and P. Norvig. *Artificial Intelligence : a modern approach*. Prentice-Hall, 1995.
- [SB98] R. Sutton and A. Barto. *Reinforcement Learning : an introduction*. MIT Press, 1998.
- [Sch92] R. Schalkoff. *Pattern Recognition*. Wiley, 1992.
- [SD90] J. Shavlik and T. Dietterich. *Readings in Machine Learning*. Morgan Kaufmann, 1990.
- [Sim85] J.-C. Simon. *La reconnaissance des formes par algorithmes*. Masson, 1985.
- [Vap98] V. Vapnik. *Statistical learning theory*. Wiley-InterScience, 1998.
- [Web99] A. Webb. *Statistical pattern recognition*. Arnold, 1999.
- [WF99] I. Witten and E. Frank. *Data Mining : Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 1999.
- [WK91] S. Weiss and C. Kulikowski. *Computer Systems That Learn*. Morgan Kaufmann, 1991.

Liste complète des références citées

- [AB89] J. Anlauf and M. Biehl. The adatron: an adaptive perceptron algorithm. *Europhysics letters*, 10:687–692, 1989.
- [AB96] M. Anthony and P. Bartlett. *Neural network learning : theoretical foundations*. Cambridge University Press, 1996.
- [Aha97] D. Aha, editor. *Lazy learning*. Kluwer, 1997.
- [Alb75] J. Albus. A new approach to manipulator control: Cerebellar model articulation controller (CMAC). *Journal of Dynamic Systems, Measurement and Control*, 97:220–227, 1975.
- [Alb78] A. Albert. *Quelques apports nouveaux à l'analyse discriminante*. PhD thesis, Faculté des sciences, Université de Liège, 1978.
- [Alb81] J. Albus. *Brains, Behavior and Robotics*. BYTE Books, McGraw-Hill, 1981.
- [AM97] N. Abe and H. Mamitsuka. Predicting protein secondary structure using stochastic tree grammars. *Machine Learning*, 29:275–301, 1997.
- [Ama68] S. Amarel. *On representation of problems of reasoning about actions*, volume 3, pages 131–171. Edinburgh University Press, 1968. Reprinted in Webber, B. and Nilsson, N. (eds.), Readings in artificial intelligence, Tioga, 1981, pp.2-22.
- [AMN94] H. Ahonen, H. Mannila, and E. Nikunen. Forming grammars for structured documents: An application of grammatical inference. In *Second International Colloquium on Grammatical Inference and Applications, ICGI'94*, volume 862 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, 1994.
- [AMS⁺95] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. Verkamo. *Fast discovery of association rules*. AAAI/MIT Press, 1995.
- [And82] J. Anderson. Logistic discrimination. In *Handbook of statistics, Vol.2, Classification, pattern recognition and reduction of dimensionality*, chapter 7, pages 169–191. North-Holland, 1982.
- [And92] D. Andler, editor. *Introduction aux sciences cognitives*. Folio. Essais, 1992.
- [Ang78b] D. Angluin. On the complexity of minimum inference of regular sets. *Information and Control*, 29(3):741–765, 1978.
- [Ang82a] D. Angluin. Inference of reversible langages. *Communications of the ACM*, 29:741–765, 1982.
- [Ang88] D. Angluin. Queries and concept learning. *Machine Learning journal*, 2:319–342, 1988.
- [Ang98] E. Angel. *La rugosité des paysages : une théorie pour la difficulté des problèmes d'optimisation combinatoire relativement aux méta-heuristiques*. PhD thesis, Laboratoire de Recherche en Informatique (LRI) de l'université Paris-Orsay, 1998.
- [AP93] P. Angeline and J. Pollack. Competitive environments evolve better solutions for complex tasks. In S. Forrest, editor, *Proc. of the 5th Int. Conf. on Genetic Algorithms, ICGA-93*, pages 264–270. Morgan-Kaufmann, 1993.
- [AU72] A. Aho and J. Ullman. *The theory of Parsing, Translation and Compiling, Vol 1 : Parsing*. Prentice-Hall, 1972.
- [AW01] K. Azoury and M. Warmuth. Relative loss bounds for on-line density estimation with the exponential family of distributions. *Machine Learning journal*, 43:211–246, 2001.
- [Axe97] R. Axelrod, editor. *The complexity of cooperation*. Princeton University Press, 1997.
- [BA97] L. Brelow and D. Aha. Simplifying decision trees: a survey. *The Knowledge Engineering Review*, 12(1):1–40, 1997.
- [Bak75] J. Baker. Stochastic modeling for automatic speech understanding. In R. Reddy, editor, *Speech Recognition*, pages 512–542. Academic Press, New York, 1975.
- [Bar91] A. Barron. *Complexity regularization with applications to artificial neural networks*, chapter Nonparametric functional estimation and related topics, pages 561–576. Kluwer, 1991.
- [Bau72] L. Baum. An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. *Inequalities*, 3:1–8, 1972.
- [Bau89] E. Baum. A proposal for more powerful learning algorithms. *Neural Computation*, 1:201–207, 1989.

- [Bax00] J. Baxter. Reinforcement learning in pomdp's via direct gradient ascent. In *17th International Conference on Machine Learning (ICML'2000)*, pages 41–48. Morgan Kaufmann, 2000.
- [BB01] P. Baldi and S. Brunak. *Bioinformatics : the machine learning approach*. MIT Press, 2001.
- [BBM96] A. Barron, L. Birgé, and P. Massart. Risk bounds for model selection via penalization. *Probability Theory Related Fields*, 1996.
- [BC91] A. Barron and T. Cover. Minimum complexity density estimation. *IEEE Transactions on Information Theory*, 37:1034–1054, 1991.
- [BC97] A. Brazma and K. Cerans. Noise-tolerant efficient inductive synthesis of regular expressions from good examples. *New Generation Computing*, 1997.
- [BEHW89] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the Association of Computer Machinery*, 36:929–965, 1989.
- [Bel61] R. Bellman. *Adaptive Control Processes : A guided tour*. Princeton University Press, 1961.
- [Ber00] M. Bernard. *Induction de programmes logiques*. Laboratoire EURISE, Université de St-Etienne, 2000.
- [BF72a] A. Biermann and J. Feldman. On the synthesis of finite-state machines from samples of their behaviour. *IEEE Transactions on Computers*, C-21:592–597, 1972.
- [BF85] D. Berry and B. Fristedt. *Bandit problems : sequential allocation of experiments*. Chapman and Hall, London, UK, 1985.
- [BGH89] L. Booker, D. Goldberg, and J. Holland. Classifier systems and genetic algorithms. *Artificial Intelligence journal*, 40:235–282, 1989.
- [Bir67] B. Birkhof. *Lattice Theory*. American Mathematical Society, 1967.
- [BKL00] A. Barberousse, M. Kistler, and P. Ludwig. *La philosophie des sciences au XXème siècle*. Flammarion. Champs Université, 2000.
- [Blu97] A. Blum. On-line algorithms in machine learning. Technical report, 1997.
- [BM94] B. Bouchon-Meunier. *La logique floue*. Presses Universitaires de France (collection Que sais-je?), 1994.
- [BM95a] J. Boyan and A. Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in Neural Information Processing Systems*, chapter 7. MIT Press, 1995.
- [BM95b] I. Bratko and S. Muggleton. Applications of Inductive Logic Programming. *Communications of the ACM*, 38(11):65–70, 1995.
- [BM98] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proc. of the 11th Annual Conf. on Computational Learning Theory, COLT'98*, pages 92–100. Morgan Kaufmann, 1998.
- [BM99] L. Baird and A. Moore. *Gradient descent for general reinforcement learning*, volume 11. MIT Press, 1999.
- [BPS94] A. Borgida and P. Patel-Schneider. Complete algorithm for subsomption in the classic description logic. *Artificial Intelligence Research*, 1:278–308, 1994.
- [BS97] A. Belaïd and G. Saon. Utilisation des processus markoviens en reconnaissance de l'écriture. *Revue Traitement du Signal*, 14,2:161–177, 1997.
- [BSS92] M. Bazaraa, D. Sherali, and C. Shetty, editors. *Non linear programming : Theory and algorithms*. Wiley-Interscience, 1992.
- [BT96] D. Bertsekas and J. Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, 1996.
- [BTC96] A. Brunie-Taton and A. Cornuéjols. Classification en programmation génétique. In *Journées Francophones d'Apprentissage (JFA-96)*, pages 303–316, 1996.
- [BTW00] J. Baxter, A. Tridgell, and L. Weaver. Learning to play chess using temporal-differences. *Machine Learning Journal*, 40 (3):243–263, 2000.
- [BU92] C. Brodley and P. Utgoff. Multivariate decision trees. Technical report, COINS, Department of Computer Science, University of Massachusetts, Amherst, USA, December 1992.

- [Bun88] W. Buntine. Generalized subsumption and its application to induction and redundancy. *Artificial Intelligence*, 36:149–176, 1988.
- [Car97] R. Carrasco. Accurate computation of the relative entropy between stochastic regular grammars. <http://www.dlsi.ua.es/~carrasco/rcc.html>, 1997.
- [CAVR98] P. Cruz-Alcázar and E. Vidal-Ruiz. Learning regular grammars to model musical style: Comparing different coding schemes. In V. Honavar and G. Slutsky, editors, *Grammatical Inference, ICGI'98*, volume 1433 of *Lecture Notes in Artificial Intelligence*, pages 211–222. Springer Verlag, 1998.
- [CBFH⁺97] N. Cesa-Bianchi, Y. Freund, D. Haussler, R. Shapire, and M. Warmuth. How to use expert advice. *Journal of American Computing Machines*, 44(3):427–485, 1997.
- [CBL01] N. Cesa-Bianchi and G. Lugosi. Worst-case bounds for the logarithmic loss of predictors. *Machine Learning journal*, 43:247–264, 2001.
- [Cel89] G. Celeux. *Classification automatique des données*. Dunod, 1989.
- [Cer85] V. Cerny. A thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.
- [Cer95] R. Cerf. An asymptotic theory for genetic algorithms. In Alliot, Lutton, Ronald, Schoenauer, and Snyers, editors, *Proceedings of the European Conference on Artificial Evolution (AE95)*, pages 37–53. Springer Verlag, LNCS-1063, 1995.
- [CGH96] I. Charon, A. Germa, and O. Hudry. *Méthodes d'optimisation combinatoire*. Masson, 1996.
- [CGV94] A. Castellanos, I. Galiano, and E. Vidal. Application of OSTIA to machine translation tasks. In *Grammatical Inference and Applications, ICGI'94*, number 862 in *Lecture Notes in Artificial Intelligence*, pages 93–105. Springer Verlag, 1994.
- [CH67] T. Cover and P. Hart. Nearest neighbour pattern classification. *IEEE Transactions on Information Theory*, 13(1), 1967.
- [Che01] Y. Chevaleyre. *Apprentissage de règles à partir de données multi-instances*. PhD thesis, Laboratoire d'informatique de Paris 6, 2001.
- [CM57a] N. Chomsky and G. Miller. Pattern conception. Technical report, ASTIA Document N° AD110076, 1957. Report N° AFCRC-TN-57-57.
- [CM99] J. Chodorowski and L. Miclet. Apprentissage et évaluation de modèles de langage par des techniques de correction d'erreurs. In *Actes de la 6ème conférence annuelle sur le Traitement Automatique des Langues Naturelles*, pages 253–262, 1999.
- [CO94] R. Carrasco and J. Oncina. Learning stochastic regular grammars by means of a state merging method. *ICGI'94*, Lecture Notes in Artificial Intelligence 862:139–152, 1994. Subseries of *Lectures Notes in Computer Science*.
- [Cor96] A. Cornuéjols. Analogie, principe d'économie et complexité algorithmique. In *Journées Francophones d'Apprentissage (JFA-96)*, pages 233–247, 1996.
- [Cov91] T. Cover. Universal portfolios. *Mathematical Finance*, 1(1):1–29, 1991.
- [CR71] S. Crespi-Reghizzi. An effective model for grammatical inference. *Proceedings of the IFIP Congress*, pages 524–529, 1971. North-Holland.
- [Cra85] N. Cramer. A representation for the adaptative generation of simple sequential programs. In *Proceedings of the International Conference on Genetic Algorithms and the Applications*, pages 183–187. Carnegie-Mellon University, Pittsburgh, PA, 1985.
- [Cre97] D. Crevier. *A la recherche de l'intelligence artificielle*. Flammarion. Champs, 1997.
- [CTC00] A. Cornuéjols, A. Tiberghien, and G. Collet. A new mechanism for transfer between conceptual domains in scientific discovery and education. *Foundations of Science*, 5(2):129–155, 2000.
- [Dar59] C. Darwin. *On the origin of species*. London: Murray, 1859.
- [Das90] B. Dasarathy, editor. *Nearest neighbor (NN) norms : NN Pattern Classification Techniques*. IEEE Computer Society Press, 1990.
- [DB93] L. De Raedt and M. Bruynooghe. A theory of clausal discovery. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 1993.

- [DB95] T. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.
- [DD97] L. De Raedt and L. Dehaspe. Clausal Discovery. *Machine Learning*, 26:99–146, 1997.
- [DDG96] F. Denis, C. D’Halluin, and R. Gilleron. PAC learning with simple examples. In *13th Symposium on Theoretical Aspects of Computer Science, STACS’96*, Lecture Notes in Computer Science, pages 231–242, 1996.
- [De 92] L. De Raedt. *Interactive Theory Revision: an Inductive Logic Programming Approach*. Academic Press, 1992.
- [Del92] J-P. Delahaye. L’altruisme recomposé? *Pour la Science*, 181, 1992.
- [DF97a] T. Dietterich and N. Flann. Explanation-based learning and reinforcement learning: a unified view. *Machine Learning journal*, 28:169–210, 1997.
- [DG97] F. Denis and R. Gilleron. PAC learning under helpful distributions. In *Proceedings of the congress on Algorithmic Learning Theory, ALT’97*, 1997.
- [DGL96] L. Devroye, L. Györfi, and G. Lugosi. *A probabilistic theory of pattern recognition*. Springer Verlag, 1996.
- [Die97] T. Dietterich. Statistical tests for comparing supervised classification learning algorithms. <http://www.cs.orst.edu/tgd/projects/supervised.html>, 1997.
- [Die00] T. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.
- [DK82] P. Devijver and J. Kittler. *Pattern Recognition: a statistical approach*. Prentice-Hall, 1982.
- [DLLP97] T. Dietterich, R. Lathrop, and T. Lozano-Pérez. Solving the multi-instance problem with axis-parallel rectangles. *Artificial Intelligence journal*, 89:31–71, 1997.
- [DM92] B. Dolsak and S. Muggleton. The application of inductive logic programming to finite element mesh design. In S. Muggleton, editor, *Inductive Logic Programming*. Academic Press, 1992.
- [DM98] P. Dupont and L. Miclet. Inférence grammaticale régulière: fondements théoriques et principaux algorithmes. Technical report, INRIA N° 3449, 1998.
- [DMV94] P. Dupont, L. Miclet, and E. Vidal. What is the search space of the regular inference? In *Grammatical inference and applications, Second International Colloquium on Grammatical Inference, ICGI’94*, volume 862 of *Lecture Notes in Artificial Intelligence*, pages 25–37. Springer-Verlag, 1994.
- [DP97] P. Domingos and M. Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning journal*, 29:103–130, 1997.
- [Dup96] P. Dupont. *Utilisation et apprentissage de modèles de langages pour la reconnaissance de la parole continue*. PhD thesis, ENST, 1996.
- [DVD96] L. Dehaspe, W. Van Laer, and L. De Raedt. Claudien the discovery engine user’s guide 3.0. Technical Report CW 239, K.U. Leuven, September 1996.
- [Dze93] S. Dzeroski. Handling imperfect data in Inductive Logic Programming. In *Proceedings of Fourth Scandinavian Conference on Artificial Intelligence*, pages 111–125. IOS Press, 1993.
- [EdB01] A. Engel and C. Van den Broeck. *Statistical mechanics of learning*. Cambridge University Press, 2001.
- [EH81] B. Everitt and D. Hand. *Finite mixture distributions*. Chapman and Hall, 1981.
- [EMS97] F. Esposito, D. Malerba, and G. Semeraro. A comparative analysis of methods for pruning decision trees. *IEEE Trans. on Patterni Analysis and Machine Intelligence*, 19(5):476–493, 1997.
- [Eng96] P. Engel. *Philosophie et psychologie*. Folio. Essais, 1996.
- [FG94] D. Foster and E. George. The risk inflation criterion for regression. *Annals of statistics*, pages 1947–1975, 1994.
- [FL94] W. Furman and B. Lindsay. Testing for the number of components in a mixture of normal distributions using moment estimators. *Computational Statistics and Data Analysis*, 17:473–492, 1994.

- [Fle88] R. Fletcher, editor. *Practical methods of optimization*. Wiley, 1988.
- [FN71] R. Fikes and N. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence journal*, 2:189–208, 1971.
- [FN72] R. Fikes and N. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence journal*, 3:251–288, 1972.
- [Fog99] L. Fogel. *Intelligence through simulated evolution, forty years of evolutionary programming*. Wiley, 1999.
- [FOW66] L. Fogel, A. Owens, and M. Walsh. *Artificial intelligence through simulated evolution*. Wiley, 1966.
- [Fre99] Y. Freund. An adaptive version of the boost by majority algorithm. In *Proc. of the 12th Annual Conf. on Computational Learning Theory, COLT'99*, pages 102–113. Morgan Kaufmann, 1999.
- [FS97] Y. Freund and R. Shapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and Systems Sciences*, 55(1):325–332, 1997.
- [FS99] Y. Freund and R. Shapire. A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 14(5):771–780, 1999.
- [FT98] G. Fauconnier and M. Turner. Conceptual integration networks. *Cognitive Science*, 22(2):133–187, 1998.
- [Fu74] K. Fu. *Syntactic Methods in Pattern Recognition*. Academic Press, 1974.
- [Gan93] J-G. Ganascia. Tdis: an algebraic formalization. In *International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 1008–1013, 1993.
- [Gar88] E. Gardner. The space of interactions in neural network models. *Journal of Physics*, A21:257–270, 1988.
- [GBD92] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–58, 1992.
- [Gin38] C.W. Gini. Variabilità e mutabilità, contributo allo studio delle distribuzioni e relazioni statistiche. Technical report, Studi Economici-Giuridici, Università di Cagliari, 1938.
- [Gio93] J-Y. Giordano. Espace des versions et inférence de grammaires algébriques. In *Journées Apprentissage, Validation et Acquisition*, 1993. Saint Raphaël, France.
- [Gir00] G. Giraud, editor. *La théorie des jeux*. Flammarion. Champs Université, 2000.
- [GJP95] F. Girosi, M. Jones, and T. Poggio. Regularization theory and neural networks architectures. *Neural Computation*, 7:219–269, 1995.
- [GK95] S. Goldman and M. Kearns. On the complexity of teaching. *Journal of Computer and System Sciences*, 50:20–31, 1995.
- [Glo86] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13:533–549, 1986.
- [Glo89a] F. Glover. Tabu search part.1. *ORSA Journal on Computing*, 1(3):190–206, 1989.
- [Glo89b] F. Glover. Tabu search part.2. *ORSA Journal on Computing*, 2(1):4–32, 1989.
- [GN88] M. Genesereth and N. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, 1988.
- [Gol67b] E. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.
- [Gol78b] M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37:302–320, 1978.
- [Gol96] R. Golden. *Mathematical methods for neural network analysis and design*. MIT Press, 1996.
- [Gre94a] J. Gregor. Data-driven inductive inference of finite-state automata. *IJPRAI*, 8,1:305–322, 1994.
- [GS00] A. Giordana and L. Saitta. Phase transitions in relational learning. *Machine Learning journal*, 41:217–251, 2000.
- [GT78] R. Gonzales and M. Thomason. *Syntactic Pattern Recognition : An introduction*. Addison-Wesley, 1978.

- [GU00] G. Grudic and L. Ungar. Localizing policy gradient estimates to action transitions. In *17th International Conference on Machine Learning (ICML'00)*, pages 343–350. Morgan Kaufmann, 2000.
- [Hau88] D. Haussler. Quantifying inductive bias: AI learning algorithms and Valiant learning framework. *Artificial Intelligence Journal*, 36:177–221, 1988.
- [Hau92] D. Haussler. Decision theoretic generalization of the pac model for neural net and other learning applications. *Information and Computation*, 100:78–150, 1992.
- [HB88] F. Van Harmelen and A. Bundy. Explanation-based generalization = partial evaluation. *Artificial Intelligence journal*, 36:401–412, 1988.
- [Her02] R. Herbrich. *Learning classifiers. Theory and beyond*. MIT Press, 2002.
- [Hil92] W. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. In *Artificial Life II*, pages 313–324. Addison-Wesley, 1992.
- [Hin89] G. Hinton. Connectionist learning procedures. *Artificial Intelligence*, 40, 1989.
- [Hir90] H. Hirsh. *Incremental Version-Space merging: a general framework for concept learning*. Kluwer, 1990.
- [Hir92] H. Hirsh. Polynomial-time learning with version spaces. In *National Conference on Artificial Intelligence*, pages 117–122, 1992.
- [HK01] J. Han and M. Kamber. *Data Mining : Concepts and Techniques*. Morgan-Kaufmann, 2001.
- [HKO01] Hyvarinen, Karhunen, and Oja. *Independent component analysis (Adaptive and learning systems for signal processing, communications and control series)*. John Wiley and Sons, 2001.
- [HKP91] J. Hertz, A. Krogh, and R. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, 1991.
- [HKS94] D. Haussler, M. Kearns, and R. Shapire. Bounds on the sample complexity of bayesian learning using information theory and the VC dimension. *Machine Learning Journal*, 14:83–113, 1994.
- [HKST96] D. Haussler, M. Kearns, H.S. Seung, and N. Tishby. Rigorous learning curve bounds from statistical mechanics. *Machine Learning Journal*, 25:195–236, 1996.
- [Hoe56] W. Hoeffding. On the distribution of the number of successes in independent trials. *Annals of Mathematical Statistics*, 27:713, 1956.
- [Hoe63a] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.
- [Hof95] D. Hofstadter. *Fluid concepts and creative analogies. Computer models of the fundamental mechanisms of thought*. Basic Books, 1995.
- [Hol75] J. Holland. *Adaptation in natural and artificial systems*. University of Michigan Press, 1975. Reprinted in 1992 by MIT Press.
- [HTF01] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning. Data mining, inference, and prediction*. Springer Verlag, 2001.
- [Jac01] Ch. Jacob. *Illustrating evolutionary computation with Mathematica*. Morgan Kaufmann, 2001.
- [Jam89] M. Jambu. *Exploration informatique et statistique des données*. Dunod, 1989.
- [Jan94] J. Jannink. Cracking and co-evolving randomizers. In *Advances in Genetic Programming*, pages 425–443. MIT press, 1994.
- [Jel76] F. Jelinek. Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, 64:532–556, 1976. No. 4.
- [Jel97] F. Jelinek. *Statistical Methods for Speech Recognition*, volume 112 of *Mathematics in Science and Engineering*. The MIT Press, 1997.
- [JP98] H. Juillé and J.B. Pollack. A stochastic search approach to grammar induction. In *Grammatical Inference*, number 1433 in Lecture Notes in Artificial Intelligence, pages 126–137. Springer-Verlag, 1998.
- [JW98] R. Johnson and D. Wichern. *Applied Multivariate Statistical Analysis*. Prentice Hall, 1998.

- [KA99] J. Koza and D. Andre. *Genetic programming III: Darwinian invention and problem solving*. MIT Press, 1999.
- [Kal99] L. Kallel. *Convergence des algorithmes génétiques : aspects spatiaux et temporels*. PhD thesis, École Polytechnique, France, 1999.
- [KCC87] S. Kedar-Cabelli and T. Mc Carty. Explanation-based generalization as resolution theorem proving. In *Proc. of the 4th Int. Worshop on Machine Learning, IWML-87*, pages 383–389. Morgan Kaufmann, 1987.
- [KGV83] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.
- [KLM96] L. Kaelbling, M. Littman, and A. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.
- [KMNR95] M. Kearns, Y. Mansour, A. Ng, and D. Ron. An experimental and theoretical comparison of model selection methods. In *Proceedings of the 8th Annual ACM Workshop on Computational Learning Theory*, pages 21–30. Morgan Kaufmann, 1995.
- [Koz92] J. Koza. *Genetic programming : on the programming of computers by mechanism of natural selection*. MIT Press, 1992.
- [Koz94] J. Koza. *Genetic programming II: Automatic discovery of reusable programs*. MIT Press, 1994.
- [KS94] M. Kearns and R. Schapire. Efficient distribution-free learning of probabilistic concepts. *Journal of Computer and System Science*, 48:464–497, 1994.
- [KS96] D. Koller and M. Sahami. Toward optimal feature selection. In *Proc. of the 13th International Conference on Machine Learning (ICML-96)*, pages 284–292. Morgan Kaufmann, 1996.
- [KV88] M. Kearns and L. Valiant. Learning boolean formula or finite automata is as hard as factoring. Technical report, Aiken Computational Laboratory, Harvard University, 1988.
- [KV89] M. Kearns and G. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. In *Proceedings of the 21st annual ACM symposium on Theory of computing*, pages 433–444, New York, 1989.
- [KV94] M. Kearns and L. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM*, 41(1):67–95, 1994.
- [KW92] J. Kietz and S. Wrobel. Controlling the complexity of learning in logic through syntactic and task-oriented models. In S. Muggleton, editor, *Inductive Logic Programming*. Academic Press, 1992.
- [Leb95] L. Lebart. *Statistique exploratoire multidimensionnelle*. Dunod, 1995.
- [Lee96] L. Lee. Learning of context-free grammars : a survey of the literature. Technical report, Center for Research in Computing Technology, Harvard University, Cambridge, Massachussets. TR 12-96, 1996.
- [Len78] D. Lenat. The ubiquity of discovery. *Artificial Intelligence journal*, 9:257–285, 1978.
- [Ler81] I-C. Lerman. *Classification et analyse ordinaire des données*. Dunod, 1981.
- [Lin91] L. Lin. Programming robots using reinforcement learning and teaching. In *Proceeding of the Ninth National Conference on Artificial Intelligence (AAAI'91)*, 1991.
- [LP97] K. Lang and B. Pearlmuter. Abbadingo one competition, 1997. abbadingo.cs.unm.edu.
- [LPP98] K. J. Lang, B. A. Pearlmuter, and R. A. Price. Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. *Lecture Notes in Computer Science*, 1433:1–12, 1998.
- [LRN86] J. Laird, P. Rosenbloom, and A. Newell. Chunking in soar: the anatomy of a general learning mechanism. *Machine Learning journal*, 1:11–46, 1986. Reprinted in Buchanan, B and Wilkins, D. (eds.), *Readings in knowledge acquisition and learning*, pp.518–535, Morgan Kaufmann, 1993.
- [LRS83] S. Levinson, R. Rabiner, and M. Sondhi. An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition. *Bell system Technical Journal*, 62:1035–1074, 1983.

- [LS98] M. Liquière and J. Sallantin. Structural machine learning with galois lattice and graphs. In J. Shavlik, editor, *15th International Conference on Machine Learning (ICML-98)*, pages 305–313. Morgan Kaufmann, 1998.
- [LTS89] E. Levin, N. Tishby, and S. Solla. A statistical approach to learning and generalization in layered neural networks. In *Workshop on Computational Learning Theory (COLT'89)*, pages 245–260. Morgan Kaufmann, 1989.
- [Lue84] D. Luenberger, editor. *Linear and nonlinear programming*. Addison-Wesley, 1984.
- [LV97] M. Li and P. Vitányi. *Introduction to Kolmogorov complexity and its applications (2nd edition)*. Springer-Verlag, 1997.
- [LVA⁺94] S. Lucas, E. Vidal, A. Amiri, S. Hanlon, and J.C. Amengual. A comparison of syntactic and statistical techniques for off-line ocr. *ICGI-94*, Lecture Notes in Artificial Intelligence, No 862:168–180, 1994.
- [LW86] N. Littlestone and M. Warmuth. Relating data compression and learnability. Technical report, University of California, Santa Cruz, 1986.
- [LW94] N. Littlestone and M. Warmuth. The weighted majority algorithm. *Information and Computation*, 108:212–261, 1994.
- [LZ96] G. Lugosi and K. Zeger. Concept learning using complexity regularization. *IEEE Transactions on Information Theory*, 42:48–54, 1996.
- [M.99] Jordan M., editor. *Learning in graphical models*. MIT Press, 1999.
- [MA95] A. Moore and C. Atkinson. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning Journal*, 21:199–233, 1995.
- [Man94] O. Mangasarian, editor. *Nonlinear programming*. SIAM, 1994.
- [Mat94] M. Mataric. Rewards functions for accelerating learning. In *Proc. of the 11th International Conference on Machine Learning, IWML-91*. Morgan Kaufmann, 1994.
- [MB88a] G. McLachlan and K. Basford. *Mixture models: Inference and application to clustering*. Marcel Dekker, 1988.
- [MB88b] S. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In Kaufmann, editor, *Proceedings of the Fifth International Conference on Machine Learning*, pages 339–352, 1988.
- [MB96] S. Matwin and F. Bergadano. Inductive Logic Programming. In *proceedings of ECAI'96*, Budapest, Hungary, 1996.
- [MC68] D. Michie and R. Chambers. *An experiment in adaptive control*, volume 2, pages 137–152. Edinburgh University Press, 1968.
- [MC85] G. Milligan and M. Cooper. An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, 50(2):159–179, 1985.
- [MC91] S. Mahadevan and J. Connell. Automatic programming of behavior-based robots using reinforcement learning. In *Proc. of the 9th National Conference on Artificial Intelligence, IWML-91*, 1991.
- [MC95a] R. Mooney and M. Califf. Induction of first-order decision lists: Results on learning the past tense of english verbs. *Journal of Artificial Intelligence Research*, 3:1–24, 1995.
- [MD94] S. Muggleton and L. De Raedt. Inductive Logic Programming: Theory and Methods. *Journal of Logic Programming*, 19-20:629–679, 1994.
- [MdG94] L. Miclet and C. de Gentile. Inférence grammaticale à partir d'exemples et de contre-exemples: deux algorithmes optimaux: (big et rig) et une version heuristique (brig). *Journées Acquisition, Validation, Apprentissage*, pages F1–F13, 1994. Strasbourg France.
- [Mei97] R. Meir. Performance bounds for nonlinear time series. In *Proceedings 10th Annual ACM Workshop on Computational Learning Theory*, pages 122–129, 1997.
- [MF90] S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proceedings of the First Conference on Algorithmic Learning Theory*, Tokyo, 1990.
- [MF98] N. Merhav and M. Feder. Universal prediction. In *Information theory. 50 years of discovery.*, pages 80–103. IEEE Press, 1998.

- [Mic76a] L. Miclet. Inference of regular expressions. In *Proceedings of the 3rd IJCPR.*, pages 100–105, 1976.
- [Mic80a] L. Miclet. Regular inference with a tail-clustering method. *IEEE Transactions on SMC, SMC-10*:737–743, 1980.
- [Mic90] L. Miclet. Grammatical inference. In H. Bunke and A. Sanfeliu, editors, *Syntactic and structural pattern recognition theory and applications*. World Scientific, 1990.
- [Min61] M. Minsky. Steps toward artificial intelligence. *Proceedings of the Institute of Radio Engineers*, 49:406–450, 1961.
- [Min88] S. Minton. *Learning search control knowledge: An explanation-based approach*. Kluwer Academic Publishers, 1988.
- [Min89] J. Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4:227–243, 1989.
- [Min90] S. Minton. Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence journal*, 42:363–392, 1990. Reprinted in Shavlik, J. and Dietterich, T. (eds.), *Readings in Machine Learning*, Morgan Kaufmann, 1990, pp.96-107.
- [Mit82] T. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.
- [MKKC86] T. Mitchell, R. Keller, and S. Kedar-Cabelli. Explanation-based generalization: a unifying view. *Machine Learning journal*, 1:45–80, 1986.
- [MKS94] S. Murphy, S. Kasif, and S. Saltzberg. A system for inducing oblique decision trees. *Journal of Artificial Intelligence Research*, 2:1–32, 1994.
- [MM96] D. Modha and E. Masrym. Minimum complexity regression estimation with weakly dependent observations. *IEEE Transactions on Information Theory*, 1996.
- [MM99] R. Munos and A. Moore. Variable resolution discretization for high-accuracy solutions of optimal control problems. In *16th international Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 1348–1355. Morgan Kaufmann, 1999.
- [Mod93] M. Modrzejewski. Feature selection using rough sets theory. In *Proc. of the European Conference on Machine Learning (ECML-93)*, pages 213–226. Springer Verlag (LNAI 667, 1993).
- [Moo96] R. Mooney. Inductive Logic Programming for Natural Language Processing. In *Proceedings of the Sixth International Workshop on Inductive Logic Programming*, pages 205–224, Stockholm, Sweden, August 1996.
- [MP43] W. McCulloch and W. Pitt. A logical calculus of ideas imminent in nervous activity. *Bulletin of Mathematical Biophysics*, 5, 1943.
- [MP69] M. Minsky and S. Papert. *Perceptrons*. MIT Press, 1969.
- [MS83] R. Michalski and R. Stepp. Automated construction of classifications: conceptual clustering versus numerical taxonomy. *IEEE Transactions on PAMI*, 5:396–409, 1983.
- [MS01] J.-F. Mari and R. Schott. *Probabilistic and Statistical Methods in Computer Science*. Kluwer, 2001.
- [Mug87] S. Muggleton. Duce, an oracle based approach to constructive induction. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 287–292. Morgan Kaufmann, 1987.
- [Mug95] S. Muggleton. Inverse entailment and proglol. *New Generation Computing*, 13(3-4):243–286, 1995.
- [MV95] L. Martin and C. Vrain. MULT_ICN: an empirical multiple predicate learner. In *Proceedings of International Workshop on Inductive Logic Programming*, 1995.
- [New90] A. Newell. *Unified theories of cognition*. Harvard University Press, 1990.
- [NMWM94] C. Nevill-Manning, I. Witten, and D. Maulsby. Compression by induction of hierarchical grammars. In *Data Compression Conference (DCC '94)*, 1994.
- [NN97] P. Njiwoua and E. Mephu Nguifo. Iggle: An instance-based learning system over lattice theory. In *Proceedings of the ICTAI'97*, pages 75–76, 1997.
- [OG92a] J. Oncina and P. García. Inferring regular languages in polynomial update time. *Pattern Recognition and Image Analysis*, pages 49–61, 1992.

- [OG92b] J. Oncina and P. García. Inferring regular languages in polynomial updated time. In *Pattern Recognition and Image Analysis : Selected papers from the IVth Spanish Symposium*, pages 49–61. World Scientific, 1992.
- [OH91] M. Opper and D. Haussler. Calculation of the learning curve of bayes optimal classification algorithm for learning a perceptron with noise. In *Workshop on Computational Learning Theory (COLT'91)*, pages 75–87. Morgan Kaufmann, 1991.
- [Orp92] P. Orponen. Neural networks and complexity theory. Technical report, Department of Computer Science, University of Helsinki., Helsinki, Finland, 1992.
- [OSW86] D. Osherson, M. Stob, and S. Weinstein. Systems that learn. *MIT Press*, 1986.
- [Par62] E. Parzen. On estimation of a probability density function and mode. *Annals of Mathematical Statistics*, 33(3), 1962.
- [Paw85] Z. Pawlak. Rough sets. *Int. J. of Computer and Information sciences*, 11, No.5:341–356, 1985.
- [Pea78] J. Pearl. On the connection between the complexity and credibility of inferred models. *Int. J. Gen. Syst.*, 4:255–264, 1978.
- [PH97] R. Parekh and V. Honavar. Learning DFA from simple examples. In *Workshop on Automata Induction, Grammatical Inference, and Language Acquisition, ICML-97*, 1997.
- [Pin97] S. Pinker. *How the mind works*. Penguin Books, 1997.
- [Pit89] L. Pitt. Inductive inference, dfa's and computational complexity. In K. P. Jantke, editor, *Workshop on analogical and inductive inference, AII89*, number 397 in Lecture Notes in Artificial Intelligence, pages 18–44. Springer-Verlag, 1989.
- [PK92] M. Pazzani and D. Kibler. The utility of knowledge in inductive learning. *Machine Learning*, 9(1):56–94, 1992.
- [Plo70] G. Plotkin. A note on induction generalization. *Machine Intelligence*, 5, 1970.
- [Plo71a] G. Plotkin. *Automatic methods of inductive inference*. PhD thesis, Edinburgh University, 1971.
- [Plo71b] G. Plotkin. A further note on induction generalization. *Machine Intelligence*, 6:101–124, 1971.
- [Pro97] J. Proust. *Comment l'esprit vient aux bêtes. Essai sur la représentation*. Gallimard. NRF Essais, 1997.
- [PS82] C. Papadimitriou and K. Steiglitz. *SCombinatorial optimization : algorithms and complexity*. Prentice-Hall, 1982.
- [PT95] A. Pelissier and A. Tête. *Sciences cognitives. textes fondateurs (1943-1950)*. PUF, 1995.
- [PW93] L. Pitt and M. Warmuth. The minimum consistent dfa problem cannot be approximated within any polynomial. *JACM*, 40(1):95–142, 1993.
- [QC95] J. Quinlan and R. Cameron-Jones. Induction of logic programs: Foil and related systems. *New Generation Computing*, 13(3-4):287–312, 1995.
- [QR89] J. Quinlan and R. Rivest. Inferring decision trees using the minimum description length principle. *Information and Computation*, 80:227–248, 1989.
- [Qui90] J. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.
- [Qui93] J. Quinlan. *C4.5 : Programs for Machine Learning*. Morgan Kaufmann, 1993.
- [Rab89] L. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77, No 2, 1989.
- [Ran00] J. Randlov. Shaping in reinforcement learning by changing the physics of the problem. In *17th International Conference on Machine Learning (ICML'2000)*, pages 767–774. Morgan Kaufmann, 2000.
- [Rey00] S. Reynolds. Adaptive resolution model-free reinforcement learning: decision boundary partitioning. In *17th International Conference on Machine Learning (ICML'2000)*, pages 783–790. Morgan Kaufmann, 2000.
- [RHW86] D. Rumelhart, G. Hinton, and R. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1: Foundations. Bradford Book, MIT Press, 1986.

- [Rip96] B. Ripley. *Pattern recognition and neural networks*. Cambridge University Press, 1996.
- [Ris78] J. Rissanen. Modeling by the shortest data description. *Automatica*, 14:465–471, 1978.
- [RJ93] L. Rabiner and F. Juang. *Fundamentals of Speech Recognition*. Prentice-Hall, 1993.
- [Rob65] J. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.
- [Rob97] S. Roberts. *An Introduction to Progol*. Oxford University, January 1997.
- [Ron96] A. Ronge. Genetic programs and co-evolution. Technical report, Department of Numerical Analysis and Computer Science, Stockholm University, 1996.
- [Ros62] F. Rosenblatt. *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms*. Spartan. Washington, DC, 1962.
- [Ros97] R. Rosenfeld. The EM algorithm. Technical report, Computer Science Department, Carnegie-Mellon University, Pittsburgh, February 1997.
- [RPV89a] H. Rulot, N. Prieto, and E. Vidal. Learning accurate finite-state structural models of words: the ECGI algorithm. In *ICASSP'89*, volume 1, pages 643–646, 1989.
- [Rud97] G. Rudolph. *Convergence properties of evolutionary algorithms*. PhD thesis, Ph.D. thesis, Universität, Hamburg, 1997.
- [RV88] H. Rulot and E. Vidal. An efficient algorithm for the inference of circuit-free automata. In G. Ferratè, T. Pavlidis, A. Sanfeliu, and H. Bunke, editors, *Advances in Structural and Syntactic Pattern Recognition*, pages 173–184. NATO ASI, Springer-Verlag, 1988.
- [RW84] R. Redner and H. Walker. Mixture densities, maximum likelihood and the em algorithm. *SIAM Review*, 26:195–239, 1984.
- [Sak90] Y. Sakakibara. Learning context-free grammars from structural data in polynomial time. *Theoretical Computer Science*, 76:223–242, 1990.
- [Sak97] Yasubumi Sakakibara. Recent advances of grammatical inference. *Theoretical Computer Science*, 185(1):15–45, 10 October 1997.
- [Sam59] A. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3:210–229, 1959.
- [Sam93] C. Sammut. The Origine of Inductive Logic Programming: a Prehistoric Tale. In *Proceedings of the 3th International Workshop on Inductive Logic Programming*, pages 127–148, Bled, Slovenia, April 1993.
- [Sap90] G. Saporta. *Probabilités, analyse des données et statistiques*. Technip, 1990.
- [SB97] S. Singh and D. Bertsekas. Reinforcement learning for dynamical channel allocation in cellular telephone systems. In *Advances in Neural Information Processing Systems : Proceedings of the 1996 Conference*, pages 974–980. MIT Press, 1997.
- [SBE99] B. Schölkopf, C. Burges, and A. Smola (Eds). *Advances in kernel methods. Support vector learning*. MIT Press, 1999.
- [SBSE00] A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans (Eds). *Advances in large margin classifiers*. MIT Press, 2000.
- [Sch90] R. Schapire. The strength of weak learnability. *Machine Learning journal*, 5(2):197–227, 1990.
- [Sch94] C. Schaffer. A conservation law for generalization performance. In *11th International Conference on Machine Learning (ICML'1994)*, pages 259–265. Morgan Kaufmann, 1994.
- [Seb94a] M. Sebag. A constraint-based induction algorithm. In W. Cohen and H. Hirsh, editors, *Proceedings of ICML-94*. Morgan Kaufmann, 1994.
- [Seb94b] M. Sebag. Using constraints to build version spaces. In *Proceedings of the 1994 European Conference on Machine Learning*, 1994.
- [SG98] R. Sun and L. Giles, editors. *Sequence learning*. Springer, 1998.
- [Sha50] C. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41:256–275, 1950.
- [Sha83] E. Shapiro. *Algorithmic program debugging*. MIT Press, 1983.
- [Sim81] H. Simon. *The sciences of the artificial*. MIT Press, 1981.

- [Sim94] K. Sims. Evolving 3d morphology and behavior by competition. In *Proc. Artificial Life IV*, pages 28–39. MIT Press, 1994.
- [SL91] S. Rasoul Safavian and D. Landgrebe. A survey of decision tree classifier methodology. *IEEE Transactions on SMC*, 21(3), 1991.
- [Sol59] R. Solomonoff. A new method for discovering the grammars of phrase structure languages. In *Information Processing*, pages 285–290. UNESCO, New York, 1959.
- [Sol64] R. Solomonoff. A formal theory of inductive inference, part 1 and part 2. *Information and Control*, 7:1–22, 224–254, 1964.
- [SR90] B. Smith and P. Rosenbloom. Incremental non-backtracking focussing: A polynomially bounded generalization algorithm for version space. In *National Conference on Artificial Intelligence (AAAI'90)*, pages 848–853, 1990.
- [SS02] B. Schölkopf and A. Smola. *Learning with kernels. Support vector machines, regularization, optimization, and beyond*. MIT Press, 2002.
- [SST92] S. Seung, H. Sompolinsky, and N. Tishby. Statistical mechanics of learning from examples. *Physical Review (A)*, 45:6056–6091, 1992.
- [STBWA96] J. Shawe-Taylor, P. Bartlett, R. Williamson, and M. Anthony. A framework for structural risk minimization. *Proceedings of the 9th Annual ACM Workshop on Computational Learning Theory*, pages 68–76, 1996.
- [STBWA98] J. Shawe-Taylor, P. Bartlett, R. Williamson, and M. Anthony. Structural risk minimization over data-dependent hierarchies. *IEEE Transactions on Information Theory*, 44:1926–1940, 1998.
- [Str00] M. Strens. A bayesian framework for reinforcement learning. In *17th International Conference on Machine Learning (ICML'2000)*, pages 943–950. Morgan Kaufmann, 2000.
- [Tak88] S. Takada. Grammatical inference for even linear languages based on control sets. *Information Processing Letter*, 28:193–199, 1988.
- [Tau94] B. Tausend. Representing biases for Inductive Logic Programming. In *proceedings of ECML'94*, pages 427–430, 1994.
- [TB73b] B. Trakhtenbrot and Ya. Barzdin. *Finite Automata: Behavior and Synthesis*. North Holland Pub. Comp., Amsterdam, 1973.
- [TD] F. Thollard and P. Dupont. Inférence grammaticale probabiliste utilisant la divergence de kullback-leibler et un principe de minimalité. In *CAP2000, Conférence d'apprentissage*, pages 259–275. Hermès.
- [TD99] F. Thollard and P. Dupont. Entropie relative et algorithmes d'inférence grammaticale probabiliste. Technical report, Laboratoire EURISE, Université de St-Etienne, 1999.
- [Tes92] G. Tesauro. Practical issues in temporal difference learning. *Machine Learning Journal*, 8:257–278, 1992.
- [Tes94a] G. Tesauro. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6:215–219, 1994.
- [Tes95a] G. Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–67, 1995.
- [TNC^{+re}] A. Tiberghien, A. N'Guyen, A. Cornuéjols, N. Balacheff, and M. Backer. *Quintette sur l'apprentissage*. Éditions l'Harmattan, à paraître.
- [TSM85] D. Titterington, A. Smith, and U. Makov. *Statistical analysis of finite mixture distributions*. John Wiley and Sons, 1985.
- [Tur92] A. Turing. Intelligent machinery. In D. Ince, editor, *Collected works of Alan Turing : Mechanical Intelligence*. Elsevier, 1992.
- [Utg89] P. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4(2), 1989.
- [Val84a] L. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.
- [Val84c] L. Valiant. A theory of the learnable. *Comunication of the ACM*, 27(11):1134–1142, 1984.
- [Vap82] V. Vapnik. *Estimation of dependences based on empirical data*. Springer-Verlag, 1982.

- [Vap95] V. Vapnik. *The nature of statistical learning theory*. Springer-Verlag, 1995.
- [VB87] K. Vanlehn and W. Ball. A version space approach to learning context-free grammars. *Machine Learning*, 2:9–38, 1987.
- [VC91] V. Vapnik and A. Chervonenkis. The necessary and sufficient conditions for the consistency of the method of empirical risk minimization. *Pattern Recognition and Image Analysis*, 1:284–305, 1991.
- [Vid94a] E. Vidal. Language learning, understanding and translation. *Progress and Prospects of Speech Research and Technology. Proceedings of the CRIM/FORWISS Workshop*, September 1994.
- [Vid94b] E. Vidal. New formulation and improvements of the nearest-neighbour approximation and eliminating search algorithm (asea). *Pattern Recognition Letters*, pages 1–7, January 1994.
- [Vid97] M. Vidyasagar. *A theory of learning and generalization*. Springer-Verlag, 1997.
- [VL63] V. Vapnik and A. Lerner. Pattern recognition using generalized portrait method. *Automation and Remote Control*, 1963.
- [VLC94] V. Vapnik, E. Levin, and Y. Le Cun. Measuring the VC-dimension of a learning machine. *Neural Computation*, 6:851–876, 1994.
- [Vos99] M. Vose. *The simple genetic algorithm. Foundations and theory*. MIT Press, 1999.
- [Wal77] R. Waldinger. Achieving several goals simultaneously. In *Machine Intelligence 8*. Ellis Horwood Ltd, 1977.
- [Wat85] S. Watanabe. *Pattern recognition: human and mechanical*. Wiley, 1985.
- [Wat89] C. Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, UK, 1989.
- [WB85] C. Wallace and D. Boulton. An information measure for classification. *Computing Journal*, 11:185–195, 1985.
- [Wer84] P. Werbos. *The Roots of Backpropagation*. Wiley, 1984.
- [Wid94] B. Widrow. 30 years of adaptive neural networks: Perceptron, madaline and backpropagation. *Proceedings of the IEEE*, 78(9), 1994.
- [Wig60] E. Wigner. The unreasonable effectiveness of mathematics in structuring physics. 1960.
- [Wil92a] R. Wille. Concept lattices and conceptual knowledge systems. *Computers Mathematical application*, 23:493–515, 1992.
- [Wil92b] R. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning Journal*, 8:229–256, 1992.
- [Win70] P. Winston. *Learning structural descriptions from examples*. PhD thesis, MIT, 1970. MIT Technical Report AI-TR-231.
- [Win75] P. Winston. Learning Structural Descriptions from Examples. In *The psychology of Computer Vision*. McGraw-Hill, 1975.
- [Wol92] D. Wolpert. On the connection between in-sample testing and generalization error. *Complex Systems*, 6:47–94, 1992.
- [Wol95] D. Wolpert, editor. *The mathematics of generalization*. Addison Wesley, 1995.
- [Wol97] D. Wolpert. No free lunch theorem for optimization. *IEEE Transactions on evolutionary computation*, 1 (1):467–82, 1997.
- [WRB93] T. Watkin, A. Rau, and M. Biehl. The statistical mechanics of learning a rule. *Review of Modern Physics*, 65:499–556, 1993.
- [YB98] Y. Yang and A. Barron. An asymptotic property of model selection criteria. *IEEE Transactions on Information Theory*, 1998.
- [ZD95] W. Zhang and T. Dietterich. A reinforcement learning approach to job-shop scheduling. In *International Joint Conference on Artificial Intelligence*, pages 1114–1120. Morgan Kaufmann, 1995.
- [Zel95] J. Zelle. *Using Inductive Logic Programming to Automate the Construction of Natural Language Parsers*. PhD thesis, University of Texas, August 1995.

- [Zha96] W. Zhang. *Reinforcement learning for job-shop scheduling*. PhD thesis, Ph.D. thesis, Oregon State University. Technical Report CS-96-30-1, 1996.
- [ZM93] J. Zelle and R. Mooney. Learning Semantic Grammars with Constructive Inductive Logic Programming. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 817–822, Washington, D.C., USA, August 1993.
- [ZR00] D. A. Zighed and R. Rakotomalala. *Graphes d'induction : apprentissage et data mining*. Hermès, 2000.
- [Zuc01] J-D. Zucker. Changements de représentation, abstractions et apprentissages. dossier d'habilitation à diriger des recherches (hdr). Technical report, 2001.

Index

- χ^2 (critère du), 340
 θ -subsomption, 163, 165, 170
 θ -subsomption relative, 172
absorbtion, 175
abstraction, 201–202
action de l’agent, 488
ADABOOST, 357
algorithme AESA, 446
algorithme BRIG, 233
algorithme d’élimination des candidats, 142
algorithme d’escalade, 101
algorithme de Baum-Welsh, 405
algorithme de condensation, 444
algorithme de nettoyage, 442
algorithme de PLI générique, 179
algorithme de rétropropagation du gradient de l’erreur, 325
algorithme de Viterbi, 401
algorithme ECGI, 230
algorithme EM, 403, 462, 482, 567–571
 apprentissage des HMM, 570
 mélange de gaussiennes, 571
algorithme k-RI, 229
algorithme RPNI, 234
algorithme RPNI stochastique, 242
algorithme *forward-backward*, 397
algorithme PROGOL, 185
algorithmes génétiques, 248–262
 espace génotypique, 247, 249, 251, 273
 espace phénotypique, 249
 fonction de performance, 255
 opérateurs de croisement, 252, 254
 opérateurs de mutation, 253, 254
 pression sélective, 255
 problèmes trompeurs, 261
 remplacement, 257–258
 sélection, 255–257
 sélection proportionnelle à la performance, 256
 sélection proportionnelle au rang, 256
 sélection par tournoi, 257
 systèmes de classeurs, 274–275
algorithmes par évolution simulée, 102
algorithmes rapides pour les k-plus proches voisins, 446
alphabet auxiliaire d’une grammaire, 213
alphabet terminal d’une grammaire, 213
AM *Automated Mathematician*, 265
analogie (raisonnement par), 203–204
analyse
 dans le pire cas, 49
 en cas moyen, 49
 PAC, 49–57
analyse en composantes principales communes, 84
analyse en composantes principales, 84
analyse en composantes indépendantes, 84
apprenant, 45
apprentissage
 à partir d’échecs, 199
 à partir d’explications (EBL), 194–201
 batch, 545
 actif, 15, 546–548
 avec bruit, 530
 collaboratif, 546
 dans les HMM, 402
 en ligne, 14
 faible, 355, 356, 360
 guidé, 548
 hors ligne, 14
 incrémental, 546–547
 mistake-bound learning model, 547
 relative loss bound model, 547
 using expert advice, 547
 multi-instance, 548
 non supervisé, 16
 par différences temporelles, 498–503
 par renforcement, 485–512
 par requête d’appartenance, 15
 paresseux, 450
 supervisé, 13, 16
 de concepts, 21
apprentissage PAC
 de grammaires, 223–226
arbres de décision, 90, 335–352
arbres de décision obliques, 343
arbres de régression, 352–354
ARCH, 37–39
astuce de la représentation unique, 133, 162
attribut

- arborescent, 128
- binaire, 77, 128
- catégoriel, 78
- nominal, 78, 128
- nominal arborescent, 78
- nominal hiérarchique, 78
- nominal totalement ordonné, 78
- numérique, 128
- séquentiel nominal, 79
- séquentiel numérique, 79
- automates à états finis, 262
- bagging*, 361
- bandit à deux bras, 489
- Bayes
 - analyse de, 57–64
 - classificateur naïf de, 61
 - maximum de vraisemblance, 530
 - règle de décision de, 60
 - règle du maximum de vraisemblance, 61
 - règle du maximum *a posteriori*, 61
 - risque de, 60
- Bayes point machines, 307
- Bellman (équation de), 495
- biais
 - de représentation, 24
 - inductif, 43
- biais de recherche, 180
- biais sémantique, 181
- biais syntaxique, 181
- blending*, 204
- boosting*, 354–361
- bootstrap*, 114, 357
- cartes auto-organisatrices de Kohonen, 84
- chaînes de Markov, 92
- chaînes de Markov cachées, *voir* HMM
- clause, 167
 - but, 168
 - définie, 167
 - de Horn, 167
 - unitaire, 168
- co-évolution, 275
- co-apprentissage, 481–483
- coévolution, 272
- cognitivisme, 11–13
- cohérence
 - d'une hypothèse, 53
- comité d'experts, 357
- comparaison de méthodes d'apprentissage, 119
- complexité algorithmique, *voir* complexité de Kolmogorov
- complexité computationnelle dans l'apprentissage PAC, 545
- complexité de Kolmogorov, 533
- compression d'information, 47
- compromis biais-variance, 42–45
- couverture d'une hypothèse, 130
- critère d'opérationnalité, 196
- critère de performance, *voir* performance (mesure de)
- critère du perceptron, 41
- croisement, 254
- cybernétique, 10–11
- d-séparation, 368, 374
- décision bayésienne, 47
- data mining*, *voir* fouille de données
- diagnostic, 372
- dilemme exploitation vs. exploration, 254, 489–491
- dimension de Vapnik-Chervonenkis, 517, 519
- early stopping rule*, 530
- EBL (*Explanation-Based Learning*), 194–201
- échantillon
 - d'apprentissage, 21, 42, 115
 - de test, 115
 - de validation, 115
- effet tunnel cognitif, 204
- élagage d'un arbre de décision, 344
- entropie croisée, 421
- environnement, 45
- épistasie, 261
- equivalence query*, *voir* protocole de requête d'équivalence
- ERM (Empirical Risk Minimization), *voir* principe ERM
- erreur
 - d'approximation, 43
 - d'estimation, 43
 - intrinsèque, 43
 - totale, 42, 44
- espace de représentation, 76–80
- espace des versions, 139
- espace génotypique, 247, 249, 251, 273
- espace phénotypique, 249
- évaluation des performances
 - courbe ROC, 116
 - par échantillon de test, 112
 - par la méthode du *leave-one-out*, 114
 - par validation croisée, 114
 - par *bootstrap*, 114
 - par *jackknife*, 114
- évolution simulée, 249
- exemple, 7
 - négatif, 38
 - positif, 38
- explication, 372
- exploitation, *voir* dilemme exploitation vs. exploration, 490

- exploration, *voir* dilemme exploitation vs. exploration, 490
extraction d'attributs, 81
extraction de connaissances des données, v
fat-shattering dimension, 305
fenêtres de Parzen, 435
filter methods, 84
fitness function, 249
fitness function, 255
FOIL, 182–184
fonction cible, 18
fonction coût, 46
fonction de croissance, 517
fonction de performance, 255
fonction de risque, 14
fonction de score, 379
fonction noyau, 434
fonction perte, 46
fonction sigmoïde, 315
fonctions noyaux, 299
fonctions séparatrices, 88
fouille de données, v
généralisé maximalement spécifique, 135
généralité (relation de), 130
Gini (critère de), 340
gradient, 325
 continu, 97
 discret, 99
grammaire formelle, 213
grammaire hors-contexte, 214
grammaire k-réversible, 228
grammaire régulière, 214
G-set, 139
heuristiques de contrôle, 195, 199, 200
HMM, 92, 387–409
 apprentissage des paramètres d'un, 402
 ergodique, 395
 gauche-droite, 395
 probabilité d'observation, 394
 probabilité de transition, 394
 probabilités initiales, 394
hypothèse
 cohérente, 132
 complète, 130, 132
 correcte, 130, 132
i.i.d. (tirage), 18, 42
identification, 175
instance, 77
intelligibilité, xiv
inter-construction, 175
intervalle de confiance, 113
intra-construction, 175
invention de prédicat, 176
inversion de la résolution, 174–177
jackknife, 114
k-plus-proches-voisins (règle des), 438
Karush-Kuhn-Tucker (conditions de), 296
KDD (*knowledge discovery in databases*), *voir* extraction de connaissances des données
Kullback-Leibler (distance de), 64
Lagrangien, 296
langage engendré par une grammaire, 213
langage hors-contexte, 214
langage régulier, 214
leave-one-out, 114
Lerman (critère de), 340
LEX, 148–149
logique de description, 95
logique des prédictats, 95
logique des propositions, 94
mélange de gaussiennes, 571
méthodes gloutonnes, 501
méthodes semi-paramétriques, 447
 discrimination logistique, 447
 mélange de distributions, 449
macro-opérateurs, 195, 199
MAP (maximum *a posteriori*), *voir* Bayes
marge d'un exemple, 359
MDLp, 379
MDLP (*Minimum Description Length Principle*),
 voir principe de description minimale
membership query, *voir* protocole de requête d'appartenance
minimisation du risque empirique (*ERM*), 50
minimisation du risque structurel (*SRM*), 527
minimisation du risque structurel (*SRM*), 526
modèle de Markov caché, *voir* HMM
modèles graphiques, 92
Monte Carlo (méthode de), 102, 498
mot engendré par une grammaire, 213
mots dérivés selon une grammaire, 213
multiplicateurs de Lagrange, 296, 529
mutation, 253, 254
near-miss, 39
nettoyage des données, xiv, 80
neurone formel, 314
no-free-lunch theorem, 254, 538–544
observation, 13
Occam (algorithme d'), 537
Occam (rasoir d'), 531

- ontologie, 92
opérateurs de généralisation, 136–137
opérateurs de spécialisation, 137
optimisation de performance, 15
oracle, 13, 45
- PAC, *voir* analyse
pas d'apprentissage (le), 41
perceptron, 39–41
algorithme, 41
performance (mesure de), 249
pertinence de l'*ERM*, 51
physique statistique (analyse de la), 548–551
politique, 485, 488
politique optimale, 493
prédiction, 14
prédiction de séquence, 262
prétraitement des données, 80
présentation
complète, 221
négative, 221
positive, 221
pression sélective, 255
principe de description minimale, 422, 534–536
principe de minimisation du risque empirique,
voir principe *ERM*
principe de minimisation du risque structurel, 526
principe du maximum de vraisemblance, 47
principe inductif, 46–48
principe *ERM*, 47
problèmes trompeurs, 261
processus de Markov, 390
PRODIGY, 200
professeur, 45
PROGOL, 184–186
programmation évolutive, 248
programmation génétique, 248, 262–272
proie-prédateur, 273
protocole d'apprentissage, 14
protocole de requête d'appartenance, 547
protocole de requête d'équivalence, 548
protocole de requête statistique, 548
- Q-learning*, 502
- régression
aux moindres carrés, 420
régression (par SVM), 307–309
régression de but, 197
régularisation (théorie de la), 527–530
réseaux bayésiens, 92
réseaux connexionnistes, 311–332
apprentissage, 319–330
architecture multicouche, 315
- choix de l'architecture, 331
exemple de fonctionnement, 316
rétropropagation du gradient de l'erreur, 325, 559–562
règle de maximum de vraisemblance, *voir* Bayes
règle de production d'une grammaire, 213
règle du maximum *a posteriori* (MAP), *voir* Bayes
règles de contrôle, 195
reconnaissance des formes, v
recuit simulé, 101
reformulation, 202
remplacement, 257–258
renforcement (signal de), 487, 488
représentation par attribut-valeur, 94
réseaux bayésiens
apprentissage des, 376–383
d-séparation, 368, 374
diagnostic, 372
explication, 372
fonction de score, 379
méthode des arbres de jonction, 373
méthode EM, 381
méthodes de conditionnement, 373
méthodes de Monte Carlo, 373
raisonnement dans les, 370–375
réseaux d'inférence bayésien, 366–370
risque empirique, 47
risque réel, 46
robot simulé, 269–271
ROC (courbe), 116
rough sets, 83
- sélection, 255–257
sélection proportionnelle à la performance, 256
sélection proportionnelle au rang, 256
sélection (étape de), 255
sélection d'attributs, 81
enveloppage, 84
filtrage, 84
sélection de modèles, 67–68
sélection par tournoi, 257
 séparateur linéaire de Bayes, 307
SARSA, 500–501
sciences de l'artificiel, 4
single representation trick, 39, 133
single representation trick, 162
SLD-résolution, 169
SLD-subsomption, 174
SOAR, 200
Solomonoff (induction selon), 532
sortie désirée, 13
spécialisé maximalement général, 135
speed-up learning, 198
S-set, 138

- statistical query model, *voir* protocole de requête statistique
stratégies d'évolution, 248, 262
subsumption (relation de), 163
superviseur, 45
SVM, 293–309, 359
 exemples critiques, 296
 fat-shattering dimension, 305
 fonctions noyaux, 299
 Lagrangien, 296
 multiplicateurs de Lagrange, 296
 théorème de Mercer, 300
 variables duales, 296
 variables ressort, 297
système apprenant, 13
systèmes de classeurs, 274, 275
systèmes dynamiques, 551

tâche d'apprentissage, 46
tabou (méthode de recherche), 102
taux de couverture, 119
 $\text{TD}(\lambda)$, 502–503
théorème de Mercer, 300
théorème des schémas, 258–262
théorie de l'estimation bayésienne, 530–531
trace d'éligibilité, 502
transition de phase, 188–190
treillis, 135

utilité
 fonction d', 492
 problème de l', 201

validation croisée, 114
variables duales, 296
variables latentes, 381
vecteurs de support,
 see exemples critiques 296
vraisemblance, 530

weight decay, 530
WINNOW, 83
wrapper methods, 84