Université de Montréal

Utilisation d'hyper-paramètres pour la sélection de variables

par

Simon Latendresse

Département d'informatique et de recherche opérationnelle

Faculté d'arts et sciences

Mémoire présenté à la Faculté des études supérieures

en vue de l'obtention du grade de

Maîtrise

en Informatique

septembre 1999

Université de Montréal

Faculté des études supérieures

Ce mémoire intitulé:

Utilisation d'hyper-paramètres pour la sélection de variables

présenté par:

Simon Latendresse

a été évalué par un jury composé des personnes suivantes:

Jean Meunier

(président-rapporteur)

Yoshua Bengio

(directeur de recherche)

Pierre L'Ecuyer

(membre du jury)

Mémoire accepté le:

15 octobre 1999

Sommaire

Ce mémoire étudie le problème de la sélection de variables pour les problèmes de régression linéaire. Etant donné un ensemble de points constitués d'un vecteur d'entrée $\mathbf{x}$ et d'une sortie désirée y, la régression linéaire consiste à trouver la meilleure droite ou le meilleur hyperplan qui explique la relation entre l'entrée et la sortie désirée. La sélection de variables consiste quant à elle à choisir parmi les coefficients du vecteur $\mathbf{x}$ ceux qui sont le plus utiles à la régression, et à éliminer ceux qui n'apportent pas suffisamment d'information utile au problème.

Nous étudions à la fois la méthode classique de résolution du problème ainsi que plusieurs nouveaux algorithmes développés pour en améliorer la performance. Les algorithmes étudiés sont la régression pas à pas, la régression ridge adaptative et un algorithme récent basé sur l'utilisation d'hyper-paramètres. Nous nous penchons en particulier sur cette dernière approche.

L'utilisation d'hyper-paramètres pour résoudre ce problème n'est pas nouvelle : la régression ridge adaptative par exemple utilise déjà cette stratégie. L'approche que nous étudions comporte toutefois deux points innovateurs. Tout d'abord, elle utilise un grand nombre d'hyper-paramètres, tandis que les autres algorithmes développés jusqu'ici en utilisent habituellement qu'un seul. Deuxièmement, alors que pour la plupart des algorithmes la valeur des hyper-paramètres doit être déterminée par essai et erreur, pour cette approche les hyper-paramètres peuvent être optimisés à l'aide d'une méthode à base de gradient.

La dernière partie de ce mémoire compare l'approche étudiée aux autres algorithmes présentés dans le mémoire. Les résultats expérimentaux obtenus in-

diquent que l'algorithme de régression ridge adaptative est celui qui produit le plus régulièrement les meilleurs résultats. L'approche à base d'hyper-paramètres y est quelque peu inférieure, mais réussit tout de même à améliorer significativement la performance de l'algorithme classique. Les résultats sont analysés et des pistes de recherche additionnelles sont suggérées pour poursuivre le développement de l'algorithme.

Table des matières

1 Introduction **1**

2 Apprentissage Supervisé **6**

2.1 Définition, Objectif et Méthodologie 6

2.2 Choix de la fonction optimale 8

2.3 Evaluation de la solution 9

2.4 Estimateurs de l'erreur de généralisation 10

2.5 Capacité 11

3 Régression Linéaire **15**

3.1 Régression linéaire classique 15

3.2 Sélection de variables 17

4 Régression pas à pas **22**

4.1 Introduction 22

4.2 Hypothèses de distribution 23

4.3 Centrer les données 24

4.4 Analyse de la variance 25

4.5 Coefficient de corrélation 29

4.6 F-test et F-test partiel 29

4.7 Algorithme détaillé 31

5 Régression ridge adaptative **33**

5.1 Régression ridge 33
5.1.1 Algorithme 33
5.1.2 Interprétation géométrique 34
5.1.3 Deuxième interprétation géométrique 36
5.1.4 Interprétation Bayesienne 39
5.2 *Non-negative garotte* et lasso 43
5.3 Régression ridge adaptative 44
5.3.1 Dérivation de l'algorithme d'optimisation 46
5.3.2 Interprétation géométrique 47
5.3.3 Interprétation Bayesienne 48
5.3.4 Détails d'implantation 49

6 Optimisation des hyper-paramètres **50**
6.1 Introduction 50
6.2 Terminologie 52
6.3 Exemples 53
6.4 Optimisation simple des hyper-paramètres 54
6.4.1 Calcul des paramètres 55
6.4.2 Gradient du critère de sélection 57
6.4.3 La décomposition de Cholesky 58
6.5 Détails d'implantation 61
6.6 Optimisation moyennée 63

7 Résultats expérimentaux **65**
7.1 Expériences préliminaires 66
7.1.1 Validation croisée *leave-one-out* 66
7.1.2 Influence de K sur la validation croisée 67
7.1.3 Estimateur 0.632 de l'erreur de généralisation 69
7.2 Comparaison des algorithmes 71
7.2.1 Structure de simulation 72

7.2.2 Expérience 1 : petit nombre d'entrées 73
7.2.3 Expérience 2 : grand nombre d'entrées 80
7.2.4 Discussion 86

8 Conclusion **89**

Chapitre 1

Introduction

Étant donné un ensemble de points constitués d'une entrée $\mathbf{x}$ et d'une sortie désirée y, la régression linéaire consiste à déterminer la meilleure droite—ou, si $\mathbf{x}$ est un vecteur, le meilleur hyperplan—expliquant la relation entre $\mathbf{x}$ et y. Il existe un algorithme classique pour résoudre ce problème : cet algorithme consiste à choisir la droite qui minimise une certaine fonction d'erreur, le coût quadratique, sur l'ensemble des points $(\mathbf{x}, y)$. La minimisation peut se faire analytiquement et le vecteur de paramètres qui définit la droite produite par l'algorithme peut être calculé simplement à partir d'une seule équation. Cet algorithme très simple éprouve toutefois certains problèmes lorsque le nombre de points disponibles pour calculer la droite est du même ordre que la taille du vecteur $\mathbf{x}$. Plusieurs algorithmes ont été créés pour tenter d'améliorer l'algorithme classique dans cette situation. Dans ce mémoire, nous proposons un tel algorithme basé sur l'utilisation d'hyper-paramètres pour pénaliser les coefficients du vecteur définissant la droite de régression. Nous comparons la performance de cet algorithme à celle de la régression linéaire classique et à la performance des autres algorithmes déjà proposés pour améliorer l'algorithme standard. Le reste de cette introduction présente une description plus détaillée du contenu du mémoire.

Le chapitre 2 fait un bref survol de la théorie de l'apprentissage et des concepts qui touchent particulièrement la régression linéaire. La régression linéaire fait par-

tie d'une catégorie de problèmes nommée apprentissage supervisé. Ces problèmes sont caractérisés par le fait que l'on dispose d'un ensemble d'exemples, nommé ensemble d'apprentissage, formés d'une entrée $\mathbf{x}$ et d'une sortie y. Le but de l'apprentissage supervisé est de trouver une relation entre $\mathbf{x}$ et y qui pourra être généralisée à d'autres exemples qui ne sont pas dans l'ensemble d'apprentissage, mais qui sont issus de la même distribution. La technique standard pour y arriver est de minimiser une certaine fonction d'erreur sur l'ensemble d'apprentissage. La fonction d'erreur que nous utilisons dans ce mémoire est le coût quadratique. La minimisation permet de sélectionner une fonction $\hat{f}$ entre les entrées et les sorties et la qualité de cette fonction est ensuite mesurée en estimant l'erreur de généralisation de cette fonction, c'est-à-dire l'espérance de l'erreur sur des points $(\mathbf{x}, y)$ tirés de la distribution inconnue d'où proviennent les exemples.

Le chapitre 3 décrit l'algorithme de régression linéaire classique. Dans le cas de la régression linéaire, la fonction $\hat{f}$ correspond à un vecteur de paramètres $\hat{\mathbf{b}}$ qui définit une droite dans l'espace des points $(\mathbf{x}, y)$. Nous décrivons l'équation du vecteur $\hat{\mathbf{b}}$ qui minimise le coût quadratique sur l'ensemble d'apprentissage. Nous discutons ensuite des heuristiques qui ont mené à l'invention de nouveaux algorithmes de régression linéaire, pour pallier aux carences de la régression classique lorsque la taille de l'ensemble d'apprentissage est trop petite.

Un des premiers algorithmes qui a ainsi été proposé est la régression pas à pas. Cet algorithme cherche à déterminer à l'aide de tests statistiques les entrées x_i qui sont les moins utiles pour calculer le vecteur $\hat{\mathbf{b}}$. Ces entrées sont éliminées du vecteur $\mathbf{x}$ et la régression classique est appliquée avec le nouveau vecteur. Réduire ainsi la taille du vecteur $\mathbf{x}$ augmente la proportion du nombre d'exemples d'apprentissage par rapport au nombre d'entrées. En général, plus le ratio entre les deux est grand, meilleure est la généralisation de l'algorithme de régression linéaire classique. La régression pas à pas est décrite au chapitre 4.

Plutôt que d'éliminer certaines entrées du vecteur $\mathbf{x}$, une autre idée est de conserver toutes les entrées, mais de pénaliser les coefficients trop grands du vec-

teur $\hat{\mathbf{b}}$. Une diminution du coefficient $\hat{b}_i$ réduit l'importance accordée à l'entrée x_i dans le processus de régression et est donc équivalent à éliminer partiellement cette entrée. Le chapitre 5 décrit deux algorithmes basés sur cette idée de pénalisation : la régression ridge et la régression ridge adaptative. La régression ridge est un algorithme datant des années soixante qui ajoute un terme de pénalité au coût quadratique. La force de la pénalité est gérée par un paramètre additionel λ que l'on nomme hyper-paramètre. La régression ridge adaptative est un algorithme récent fondé sur la même idée, mais qui utilise plusieurs termes de pénalités plutôt qu'un seul. Chaque terme de pénalité est pondéré par son propre hyper-paramètre λ_m et ces M hyper-paramètres sont soumis à une contrainte $g(\mu, \lambda_1, \ldots, \lambda_M)$ qui restreint les valeurs qu'ils peuvent adopter. Lorsque la variable μ est fixée, la valeur optimale des hyper-paramètres peut être déterminée par un processus de minimisation similaire à celui utilisé pour calculer $\hat{\mathbf{b}}$.

L'algorithme que nous proposons et que nous étudions ici est également basé sur l'idée de pénalisation. Il utilise des termes de pénalité similaires à ceux utilisés par la régression ridge adaptative, mais sans contrainte sur les hyper-paramètres λ_m. Dans le cas de la régression ridge adaptative, il n'y a à proprement parler qu'un seul hyper-paramètre, μ, puisque la valeur des λ_m est déterminée par la valeur de μ. Dans le cas de l'algorithme que nous proposons, les λ_m sont des hyper-paramètres à part entière. Pour déterminer leur valeur optimale, nous proposons d'utiliser un estimateur de l'erreur de généralisation, dérivable par rapport aux λ_m. On peut alors utiliser un algorithme d'optimisation à base de gradient pour déterminer la valeur des hyper-paramètres qui minimisera cet estimateur. La procédure est décrite en détail au chapitre 6 sous le nom d'optimisation des hyper-paramètres.

Nous avons fait plusieurs expériences pour tester les algorithmes décrits dans les chapitres 4 à 6. Nous avons d'abord testé l'utilisation de différents estimateurs de l'erreur de généralisation de concert avec l'algorithme d'optimisation des hyper-paramètres. Nous avons ensuite testé tous les algorithmes avec des données

issues de différentes distributions. La régression ridge adaptative est l'algorithme qui a produit le plus souvent les meilleurs résultats. L'algorithme d'optimisation des hyper-paramètres y est un peu inférieur : bien que dans certains cas il produit les meilleurs résultats de tous les algorithmes, sa performance est en général moins bonne que celle de la régression ridge adaptative.

Notation et nomenclature

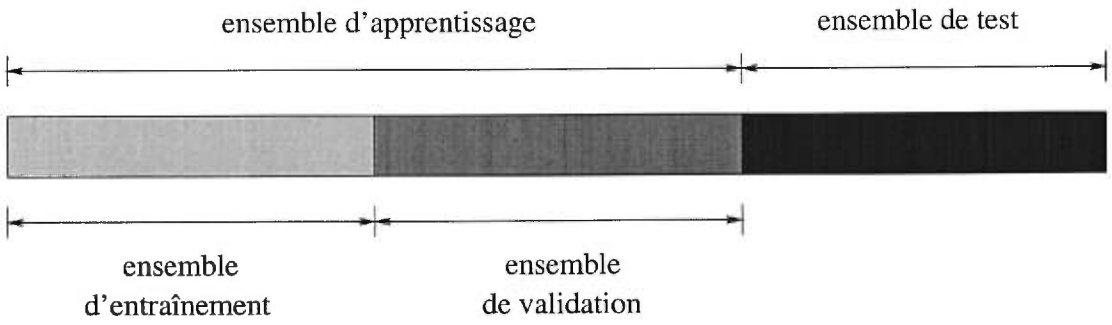
La notation utilisée dans ce mémoire obéit à un certain nombre de principes généraux. Les scalaires sont dénotés par des lettres minuscules en italique tirées de l'alphabet romain ou grec. Un exemple est la sortie désirée y utilisée plus haut. Les lettres minuscules en caractère gras dénotent les vecteurs, deux exemples étant $\mathbf{x}$ et $\boldsymbol{\gamma}$. Les coefficients des vecteurs sont des minuscules italiques indexées au coin inférieur droit : $\mathbf{x} = (x_1, \ldots, x_M)$. Les vecteurs correspondant à chaque exemple de l'ensemble d'apprentissage sont numérotés par un exposant. Les T vecteurs d'entrée sont ainsi dénotés $\mathbf{x}^1, \ldots, \mathbf{x}^T$. Pour dénoter les matrices, on utilise des lettres majuscules en caractère gras. Leurs coefficients sont en italique comme les coefficients des vecteurs : $\mathbf{H} = (H_{ij})$.

Certaines variables sont utilisées tout au long du mémoire. Elles sont définies avant leur première utilisation, mais nous prenons tout de même la peine de les mentionner ici. La taille de l'ensemble d'apprentissage est désignée par la variable T. Le nombre de coefficients de chaque vecteur d'entrée est M. Pour indexer les exemples de l'ensemble d'apprentissage, on utilise la lettre t. Dans toutes les sommes et toutes les énumérations, on sous-entend implicitement que t prend les valeurs de 1 à T. La même chose est valide pour la lettre m. Ainsi, l'équation

$$\bar{x} = \frac{1}{M} \sum_m x_m$$

fait la moyenne des M coefficients du vecteur $\mathbf{x}$. La variable $\mathbf{z}$ est utilisée pour désigner un vecteur qui est la concaténation de l'entrée $\mathbf{x}$ et de la sortie y : $\mathbf{z} = (\mathbf{x}, y)$. Finalement, l'ensemble d'apprentissage est dénoté par la lettre D.

Les termes ensemble d'apprentissage, ensemble d'entraînement, ensemble de validation et ensemble de test sont utilisés pour désigner des ensembles d'exemples bien précis. L'ensemble d'apprentissage est constitué de tous les exemples disponibles pour déterminer la droite de régression. Certains algorithmes divisent cet ensemble en deux parties : la première partie est utilisée pour calculer le vecteur **b** définissant la droite, et la deuxième pour mesurer la qualité de la solution trouvée, en général pour estimer la qualité obtenue avec une certaine valeur des hyper-paramètres. On nomme la première partie l'ensemble d'entraînement et la deuxième, ensemble de validation. L'ensemble de test est un ensemble distinct de l'ensemble d'apprentissage. Il est utilisé pour estimer l'erreur de généralisation d'un algorithme. Nous utilisons de tels ensembles dans nos expériences pour mesurer la performance de chaque algorithme sur des données qui n'ont pas servis à l'apprentissage. On peut ainsi comparer entre eux plusieurs algorithmes de façon non-biaisée. La figure suivante résume la situation :



Remarquez que la taille de chaque ensemble varie d'un algorithme à l'autre et d'une expérience à l'autre. Les tailles ne sont pas nécessairement celles indiquées par la figure.

Chapitre 2

Apprentissage Supervisé

2.1 Définition, Objectif et Méthodologie

L'objectif de l'apprentissage supervisé est d'apprendre une relation entre une **variable d'entrée** et une **variable de sortie** à partir d'un ensemble d'exemples que l'on nomme **ensemble d'apprentissage**.

Soit $D = \{\mathbf{z}^1, \ldots, \mathbf{z}^T\}$ l'ensemble d'apprentissage. Chaque élément de cet ensemble a la forme $\mathbf{z}^t = (\mathbf{x}^t, y^t)$, où $\mathbf{x}^t = (x_1^t, \ldots, x_M^t)$ est un vecteur de taille M et y^t est un scalaire. Pour chaque exemple $\mathbf{z}^t$, le vecteur $\mathbf{x}^t$ est la variable d'entrée et y^t est la variable de sortie correspondante. Les variables $\mathbf{z}$ sont issues d'une certaine distribution inconnue $p(\mathbf{z})$. La seule information disponible sur cette distribution est l'ensemble D, dont les éléments ont été tirés de $p(\mathbf{z})$. L'objectif de l'apprentissage supervisé est d'apprendre une relation $y = f(\mathbf{x})$ entre les variables d'entrées et les variables de sortie, selon un certain critère d'optimalité.

Apprendre parfaitement cette relation à partir de l'ensemble d'apprentissage est généralement impossible. Premièrement, la relation peut être bruitée, c'est-à-dire que plusieurs y différents peuvent correspondre au même $\mathbf{x}$. Deuxièmement, même si parmi l'ensemble d'apprentissage aucun $\mathbf{x}$ n'est associé à plus d'un y, le nombre de fonctions mathématiques possibles reliant les entrées aux sorties est infini. Il existe une infinité de fonction f_n telles que $y^t = f_n(\mathbf{x}^t)$ pour $t = 1, \ldots, T$.

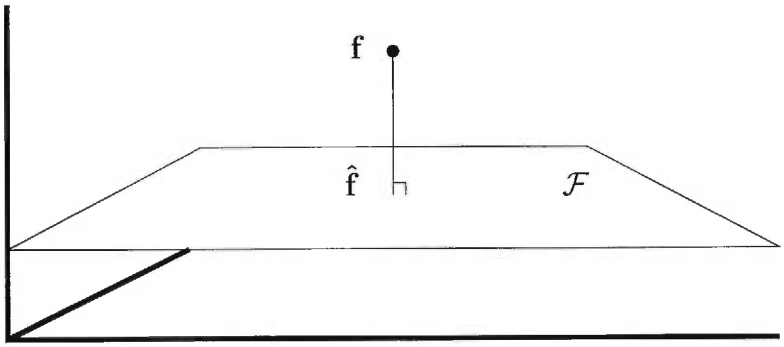


FIG. 2.1: Recherche de la fonction f. La fonction n'appartient pas à l'espace $\mathcal{F}$ auquel on restreint la recherche. La meilleure fonction $\hat{f}$ est la fonction parmi $\mathcal{F}$ qui est la plus près de f.

Chercher la véritable relation parmi cet ensemble infini de possibilités est impossible. Troisièmement, même si y était une fonction déterministe de $\mathbf{x}$, comment trouver cette fonction parmi l'ensemble infini des fonctions mathématiques ?

Pour resoudre ces problèmes, il faut restreindre le domaine de recherche : plutôt que de considérer toutes les fonctions possibles, on choisira plutôt un ensemble de fonctions $\mathcal{F}$ auquel on limitera la recherche. Soit $\hat{f}$ la fonction appartenant à $\mathcal{F}$ qui se rapproche le plus de la véritable fonction f, selon une certaine mesure de distance entre fonctions. Comme $\mathcal{F}$ ne contiendra habituellement pas f, $\hat{f}$ sera une fonction différente de f. On accepte de faire une erreur dans le choix de f, mais en échange on pourra choisir pour $\mathcal{F}$ un ensemble pour lequel il existe un algorithme qui permet de faire une recherche efficace de $\hat{f}$. Ce n'est bien sûr pas tous les ensembles qui ont cette propriété : certains ensembles ne permettent pas une telle recherche. Mais il existe tout de même plusieurs possibilités de $\mathcal{F}$. Dons notre cas, nous choisiront $\mathcal{F}$ comme étant l'ensemble des fonctions linéaires.

La figure 2.1 illustre la recherche de f. La fonction f existe quelque part dans l'espace des fonctions mathématiques, représenté dans la figure comme étant un espace en trois dimensions. L'ensemble $\mathcal{F}$ est un sous-ensemble de ce vaste espace. Nous voulons choisir comme valeur de $\hat{f}$ la fonction appartenant à $\mathcal{F}$ qui est la

plus proche de la véritable fonction f.

2.2 Choix de la fonction optimale

Pour choisir $\hat{f}$ parmi $\mathcal{F}$, on utilise une fonction $Q(f, \mathbf{z})$ qui mesure l'erreur faite sur l'exemple $\mathbf{z}$ par la fonction f. Cette fonction peut avoir différentes formes. Dans notre cas, nous utiliserons le **coût quadratique**

$$Q(f, \mathbf{z}) = \frac{1}{2}(f(\mathbf{x}) - y)^2. \tag{2.1}$$

Notre objectif est de trouver $\hat{f}$ parmi $\mathcal{F}$ qui minimise l'**erreur de généralisation**

$$R(f) = \int Q(f, \mathbf{z}) p(\mathbf{z}) d\mathbf{z}. \tag{2.2}$$

Toutefois, cette erreur ne peut être calculée car la distribution $p(\mathbf{z})$ est inconnue. Il est cependant possible d'estimer cette erreur. Considérons la fonction suivante :

$$R(f, S) = \frac{1}{|S|} \sum_{\mathbf{s}^i \in S} Q(f, \mathbf{s}^i). \tag{2.3}$$

Cette fonction mesure l'erreur moyenne de la fonction f sur les éléments d'un ensemble S. Pour un f quelconque choisi indépendamment de D, $R(f, D)$ est un estimateur bruité mais non-biaisé de $R(f)$. Le fait que l'estimateur soit non-biaisé signifie que

$$E_D[R(f, D)] = R(f). \tag{2.4}$$

Considérons les variables

$$f^*(D) = \text{argmin}_{f \in \mathcal{F}} R(f, D) \tag{2.5}$$

$$f^* = \text{argmin}_{f \in \mathcal{F}} R(f). \tag{2.6}$$

La fonction $f^*(D)$ est un estimateur de f^* et est un choix raisonnable pour $\hat{f}$. Cette façon de sélectionner $\hat{f}$ est à la base d'un grand nombre d'algorithmes d'apprentissage et porte le nom de **principe de minimisation du risque empirique**. On nomme $R(f^*(D))$, l'erreur faite sur l'ensemble d'apprentissage par la fonction sélectionnée par l'algorithme, l'**erreur d'apprentissage**.

2.3 Evaluation de la solution

In est important de noter que l'erreur d'apprentissage $R(f^*(D), D)$ n'est pas un estimateur non-biaisé de l'erreur de généralisation $R(f^*(D))$ parce que $f^*(D)$ n'est pas indépendante de D, tel que noté plus haut. L'erreur d'apprentissage de $f^*(D)$ sera plus petite que son erreur de généralisation parce que la fonction $f^*(D)$ a été choisie spécialement pour minimiser l'erreur sur l'ensemble D. Cet ensemble est un échantillon de taille finie tiré de $p(\mathbf{z})$, et la distribution empirique $\hat{p}(\mathbf{z})$ des exemples de cet échantillon n'est pas en général la même que $p(\mathbf{z})$, de la même façon que lorsqu'on tire à pile ou face avec une pièce de monnaie ordinaire, on n'obtient en général pas exactement autant de piles que de faces. Pour que la distribution des exemples d'apprentissage soit vraiment identique à $p(\mathbf{z})$, il faudrait un ensemble d'apprentissage de taille infinie. Ceci n'est pas le cas. La fonction $f^*(D)$ minimise l'erreur pour une distribution $\hat{p}(\mathbf{z})$ légèrement différente de $p(\mathbf{z})$. En conséquence, son erreur de généralisation, mesurée sur $p(\mathbf{z})$, sera plus grande que son erreur d'apprentissage, qui elle est mesurée sur $\hat{p}(\mathbf{z})$.

Donc, pour mesurer la qualité de la fonction $\hat{f} = f^*(D)$, l'erreur d'apprentissage à elle seule ne suffit pas. L'erreur d'apprentissage est utile pour sélectionner la fonction, mais il faut un autre moyen pour estimer de l'erreur de généralisation pour cette fonction. Pour y arriver, on peut utiliser l'équation 2.3 avec un ensemble D' tiré de $p(\mathbf{z})$ tel que D' est indépendant de $\hat{f}$, cette équation nous donne une estimation non-biaisé de l'erreur de généralisation.

La procédure d'entraînement d'un algorithme d'apprentissage se résume donc ainsi. Etant donné un ensemble d'exemples, on sépare cet ensemble en deux sous-ensembles distincts : un ensemble d'apprentissage D et un ensemble de test D'. On choisit la fonction $\hat{f}$ qui minimise l'erreur quadratique sur les éléments de l'ensemble d'apprentissage. On calcule ensuite l'erreur quadratique moyenne sur les éléments de l'ensemble de test, ce qui nous donne un estimateur de l'erreur de généralisation de la fonction.

2.4 Estimateurs de l'erreur de généralisation

Il existe d'autres estimateurs de l'erreur de généralisation, plus complexes que celui présenté à la section précédente. Nous utiliserons un de ces estimateurs, la **validation croisée**, dans les algorithmes présentés plus loin. La technique de la section précédente utilise mal les exemples mis à sa disposition. Elle ne se sert que d'une partie des exemples pour choisir la fonction $\hat{f}$, et que d'une partie pour estimer son erreur de généralisation. L'ensemble de test ne sert qu'à mesurer la qualité de la solution et n'est pas utilisé pendant le processus d'entraînement. Cette restriction est nécessaire car on veut évaluer la solution en utilisant des exemples qui n'ont pas servi à l'entraînement. Cependant, on remarque que l'erreur obtenue sur l'ensemble de test n'est pas simplement un indice de la qualité de la fonction $\hat{f}$ choisie, c'est aussi—et même, surtout—un indice de la qualité de l'algorithme qui a servi à obtenir cette solution. La validation croisée sépare les exemples disponibles en K paires d'ensembles distincts, $S_1^1 \cup S_2^1$, $S_1^2 \cup S_2^2$, ..., $S_1^K \cup S_2^K$. Pour chaque paire d'ensembles, elle entraîne l'algorithme sur le premier et le teste sur le deuxième. Elle fait ensuite la moyenne des estimateurs d'erreur de généralisation obtenus.

$$E = \frac{1}{K} \sum_i R(f^*(S_1^i), S_2^i). \tag{2.7}$$

Chacun de ces estimateurs est obtenu en évaluant la fonction choisie sur des exemples qui n'ont pas servi à l'entraînement. Et chacun est un estimateur de l'erreur de généralisation de l'algorithme. En faisant la moyenne, on obtient un estimateur de l'erreur de généralisation qui a été calculé en utilisant plus d'exemples que si on n'avait fait qu'une simple séparation des exemples en ensemble d'apprentissage versus ensemble de test. Il est donc raisonnable de croire que cet estimateur mesure plus précisément l'erreur de généralisation que l'on peut espérer de la part de l'algorithme quand on l'entraînera avec de telles données.

2.5 Capacité

La qualité de la fonction $\hat{f}$ produite par un algorithme d'apprentissage dépend habituellement de la taille de l'ensemble $\mathcal{F}$ où s'effectue la recherche de la solution. Plus spécifiquement, l'erreur de généralisation faite par la fonction $\hat{f}$ est une fonction d'une certaine caractéristique de l'ensemble $\mathcal{F}$, qu'on nomme la **capacité** de cet ensemble et qu'on dénote $h(\mathcal{F})$. Intuitivement, la capacité d'un ensemble $\mathcal{F}$ correspond au nombre de fonctions différentes qui y appartiennent. Plus le choix de fonctions est vaste, plus grande est la capacité de l'ensemble.

La définition formelle de la capacité est présentée par Vladimir Vapnik dans son livre "The Nature of Statistical Learning Theory" [31]. Dans le cas d'un problème de classification, elle peut être formulée simplement comme suit : la capacité $h(\mathcal{F})$ d'un ensemble $\mathcal{F}$ est le plus grand T tel qu'il existe un ensemble d'exemples D de taille T pour lequel on peut toujours trouver une fonction $f \in \mathcal{F}$ qui classifie correctement les exemples de D, pour tous les étiquetages possibles de ces exemples. Considérons un exemple en deux dimensions. Soit $\mathcal{F}_{\text{lin}}$ l'ensemble des fonctions linéaires de la forme $f(x) = mx + b$. La figure 2.2 montre qu'il existe un ensemble D de taille trois pour lequel il est possible de classifier correctement les exemples pour tout étiquetage de ces exemples. La capacité de l'ensemble $\mathcal{F}_{\text{lin}}$ est donc d'au moins trois. Si on essaie de rajouter un quatrième point à l'ensemble D, on voit qu'il est alors impossible de classifier correctement les points pour tous les étiquetages. La figure 2.3 donne un exemple d'un étiquetage qui ne peut être classifié correctement avec une ligne droite. Il est en fait impossible de trouver un ensemble de quatre points qui pourra toujours être classifié correctement avec une ligne droite pour tous les étiquetages. Il surviendra toujours un étiquetage qui, comme pour la figure 2.3, ne pourra être classifié correctement. Pour cette raison, la capacité de l'ensemble $\mathcal{F}_{\text{lin}}$ est de trois.

Pour les problèmes de régression, la définition formelle de la capacité est beaucoup plus complexe. Toutefois, intuitivement, il s'agit simplement d'une

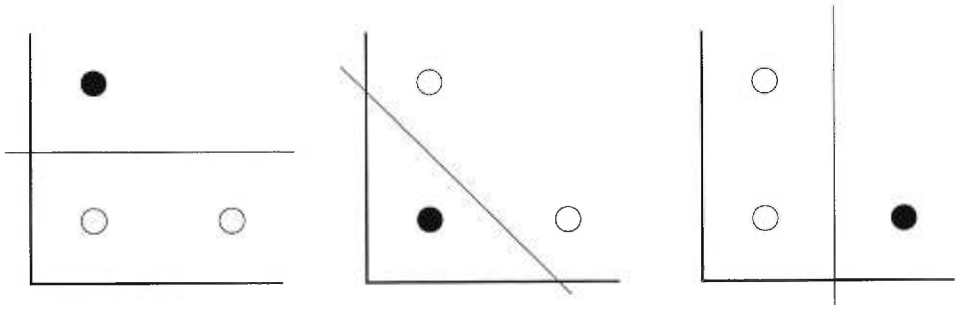


FIG. 2.2: Classification de trois exemples à l'aide d'une ligne droite. Pour tout étiquetage des trois exemples, il est possible de classifier les exemples en deux catégories.

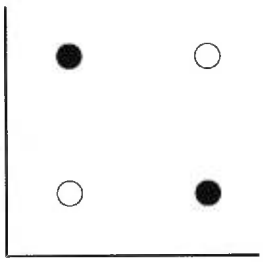


FIG. 2.3: Exemple d'un ensemble de quatre exemples qui ne peuvent être classifiés correctement à l'aide d'une ligne droite.

généralisation de la définition donnée pour les problèmes de classification. Pour un ensemble d'apprentissage D donné, un ensemble $\mathcal{F}$ de grande capacité aura tendance à produire une fonction $\hat{f}$ qui "colle" beaucoup aux exemples de l'ensemble, tandis qu'un ensemble $\mathcal{F}$ de capacité plus petite produira une fonction qui commet une plus grande erreur sur les exemples de D. La figure 2.4 illustre la situation. Un polynôme de degré deux a une plus grande capacité qu'un polynôme de degré un—i.e. une droite. Si on utilise les deux pour faire une régression sur un ensemble de points, le polynôme de degré deux commettra moins d'erreur sur chacun des exemples.

Une grande capacité signifie une petite erreur d'apprentissage, mais ceci n'implique pas nécessairement une petite erreur de généralisation. Au contraire :

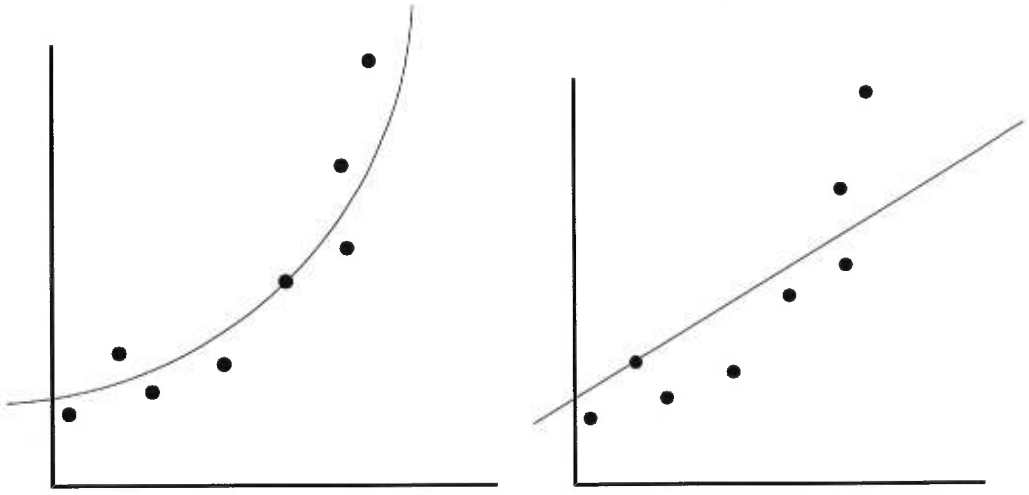


FIG. 2.4: Régression sur un ensemble de points. Un polynôme de degré supérieur, correspondant à un ensemble de fonctions de plus grande capacité, commet moins d'erreur sur l'ensemble d'apprentissage.

bien qu'il est vrai que l'erreur d'apprentissage diminue constamment à mesure que la capacité augmente, l'erreur de généralisation quant à elle va plutôt diminuer jusqu'à un certain minimum pour ensuite augmenter lorsque la capacité est trop grande. La figure 2.5 présente le graphe des erreurs d'apprentissage et de généralisation en fonction de la capacité. Lorsqu'un algorithme d'apprentissage se situe à la gauche du point optimal sur le graphique, on dit que l'algorithme ne s'ajuste pas suffisamment aux données. Dans cette situation, l'algorithme n'a pas assez bien appris la structure de l'ensemble d'apprentissage : son erreur d'apprentissage et son erreur de généralisation sont trop élevées. Lorsqu'un algorithme se trouve à droite du point optimal, on dit qu'il s'ajuste trop aux données. L'algorithme accorde alors trop d'importance à la structure particulière de l'ensemble d'apprentissage. Tel que mentionné à la section 2.3, la distribution empirique $\hat{p}(\mathbf{z})$ des exemples de l'ensemble d'apprentissage n'est pas en général la même que la distribution $p(\mathbf{z})$ de tous les exemples. Si l'algorithme choisit une fonction qui colle trop à $\hat{p}(\mathbf{z})$, son erreur d'apprentissage sera très faible, mais son erreur de généralisation sera grande car il n'aura pas su reconnaître la véritable distribution

des exemples.

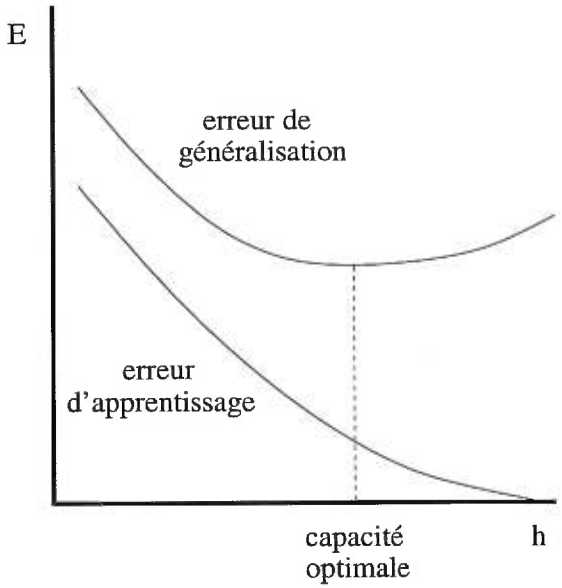


FIG. 2.5: Graphique de l'erreur d'apprentissage et de l'erreur de généralisation en fonction de la capacité.

Chapitre 3

Régression Linéaire

3.1 Régression linéaire classique

La régression linéaire est un des algorithmes les plus utilisés pour analyser des données expérimentales. Étant donné un ensemble d'apprentissage D tel que défini au chapitre précédent, on veut trouver la fonction linéaire $\hat{f}$ qui minimise l'erreur d'apprentissage $R(f, D)$. Une fonction linéaire est une fonction de la forme $f(\mathbf{x}) = b_0 + \mathbf{b}'\mathbf{x}$. Pour simplifier la notation, il est pratique d'ajouter un coefficient 1 comme premier élément du vecteur $\mathbf{x}$. On peut alors considérer que le vecteur $\mathbf{b}$ est un vecteur de taille $M+1$ et éliminer le terme b_0 de l'équation. Donc, pour le reste de ce mémoire, nous considérons qu'une fonction linéaire est une fonction de la forme

$$f(\mathbf{x}) = \mathbf{b}'\mathbf{x}. \tag{3.1}$$

Plusieurs équations en régression linéaire peuvent être simplifiées en utilisant la notation matricielle. Pour cette raison, nous combinons les T exemples de l'ensemble d'apprentissage en une matrice $\mathbf{X}$ de taille $T \times (M+1)$. De même, on combine les T sorties désirées en un vecteur $\mathbf{y} = (y^1, \ldots, y^T)$. L'équation 3.1 s'écrit alors

$$f(\mathbf{X}) = \mathbf{X}\mathbf{b}. \tag{3.2}$$

Si on veut mettre les coefficients en évidence, on peut écrire

$$f(\mathbf{X}) = \begin{pmatrix} 1 & x_1^1 & x_2^1 & \cdots & x_M^1 \\ 1 & x_1^2 & x_2^2 & \cdots & x_M^2 \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_1^T & x_2^T & \cdots & x_M^T \end{pmatrix} \cdot \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{M+1} \end{pmatrix}. \tag{3.3}$$

Comme la fonction f est complètement déterminée par le vecteur de paramètres $\mathbf{b}$ correspondant, nous utiliserons dorénavant ce vecteur pour représenter la fonction. Ainsi, nous redéfinissons le coût quadratique pour le cas linéaire

$$Q(\mathbf{b}, \mathbf{z}) = \frac{1}{2}(\mathbf{x}'\mathbf{b} - \mathbf{y})^2. \tag{3.4}$$

De même, on rédéfinit l'erreur d'apprentissage

$$R(\mathbf{b}, D) = \frac{1}{T}\sum_t Q(\mathbf{b}, \mathbf{z}^t). \tag{3.5}$$

Cette équation peut être réécrite sous forme matricielle

$$R(\mathbf{b}, D) = \frac{1}{2T}(\mathbf{X}\mathbf{b} - \mathbf{y})'(\mathbf{X}\mathbf{b} - \mathbf{y}). \tag{3.6}$$

Cette forme est plus spécialisée mais plus compacte et c'est elle que nous utiliserons dorénavant.

Comme nous l'avons déjà mentionné, le but de la régression linéaire est de trouver le vecteur de paramètres $\mathbf{b}$ qui minimise l'erreur d'apprentissge $R(\mathbf{b}, D)$. Ce minimum se trouve au point où la dérivée de R par rapport à $\mathbf{b}$ est zéro. Pour calculer ce point, réécrivons d'abord l'équation de R sous la forme

$$R(\mathbf{b}, D) = \frac{1}{2T}(\mathbf{b}'\mathbf{X}'\mathbf{X}\mathbf{b} - \mathbf{b}'\mathbf{X}'\mathbf{y} - \mathbf{y}'\mathbf{X}\mathbf{b} + \mathbf{y}'\mathbf{y}). \tag{3.7}$$

Comme les termes de cette somme sont des scalaires, on peut transposer le troisième terme et ainsi obtenir

$$R(\mathbf{b}, D) = \frac{1}{2T}(\mathbf{b}'\mathbf{X}'\mathbf{X}\mathbf{b} - 2\mathbf{b}'\mathbf{X}'\mathbf{y} + \mathbf{y}'\mathbf{y}). \tag{3.8}$$

On calcule ensuite la dérivée par rapport à $\mathbf{b}$ au point $\hat{\mathbf{b}}$ où la dérivée vaut zéro.

$$\left.\frac{\partial R}{\partial b}\right|_{\hat{\mathbf{b}}} = \frac{1}{T}(\mathbf{X}'\mathbf{X}\hat{\mathbf{b}} - \mathbf{X}'\mathbf{y}) = 0 \tag{3.9}$$

On peut donc calculer le point où R atteint son minimum. Ce point est donné par l'équation

$$\hat{\mathbf{b}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}. \tag{3.10}$$

Remarquez que pour que la solution existe, il faut que $\mathbf{X}'\mathbf{X}$ soit inversible. L'inverse existe toujours lorsque les entrées sont linéairement indépendantes, c'est-à-dire lorsque aucune colonne de la matrice $\mathbf{X}$ n'est une combinaison linéaire des autres.

3.2 Sélection de variables

Nous avons présenté un algorithme pour calculer le vecteur de paramètres $\hat{\mathbf{b}}$ pour prédire un vecteur de sorties désirées $\mathbf{y}$ en se basant sur une série de variables d'entrées, représentées par la matrice $\mathbf{X}$. On veut maintenant savoir : est-il possible de trouver un vecteur de paramètres pour lequel l'erreur de généralisation serait encore mieux ? Une approche qui a été beaucoup étudiée est de tenter de diminuer l'influence des variables d'entrée les moins significatives. Plusieurs raisons ont été avancées pour justifier cette approche.

Dans la pratique, les expérimentateurs ignorent la plupart du temps ce qui fait varier la variable de sortie mesurée. Ils tentent d'amasser le plus d'information possible sur les conditions de l'expérience, mais certaines des variables mesurées ont souvent très peu ou pas du tout d'influence sur la variable de sortie. Ils appliquent ainsi la régression à des problèmes où plusieurs des colonnes de la matrice $\mathbf{X}$ sont du bruit. C'est donc une bonne idée d'éliminer le plus de ces variables d'entrées inutiles avant de calculer le vecteur de paramètres $\hat{\mathbf{b}}$.

Ces considérations ne sont pas purement pratiques. Elles ont aussi une justification théorique. La régression linéaire classique donne de piètres résultats lorsque

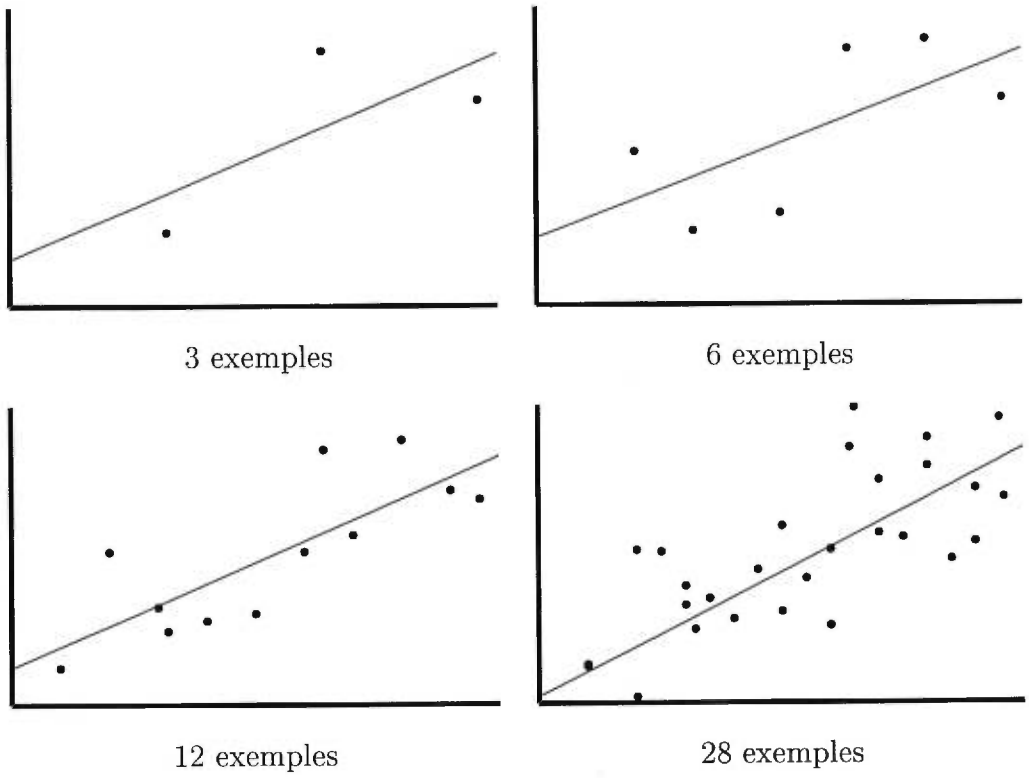


FIG. 3.1: Exemple de régression linéaire classique. Plus le nombre d'exemples d'apprentissage est grand, plus la droite estimée est proche de la véritable relation entre x et y.

le nombre de paramètres est du même ordre que la taille de l'ensemble d'apprentissage. Un exemple de cette situation est donné à la figure 3.1. Nous avons généré des points le long d'une droite à quarante-cinq degrées en ajoutant du bruit Gaussien $N(0,1)$ autour de celle-ci. Nous avons ensuite utilisé l'algorithme de régression linéaire classique pour trouver la meilleure droite passant par ces points. On voit que plus le nombre d'exemples est grand, plus il est facile de retrouver la relation entre l'entrée et la sortie. Quand il n'y a pas assez d'exemples, le bruit sur les points rend l'estimation des paramètres difficile et la droite calculée est loin de la véritable régression. On peut voir ceci comme un problème de capacité : lorsque le nombre de paramètres est proche du nombre d'exemples d'apprentissage, la capacité du système est trop grande pour les exemples dis-

ponibles et le système s'ajuste trop aux données. Tel que discuté au chapitre précédent, il existe une capacité optimale pour tout système, capacité qui produira la plus faible erreur de généralisation. Dans le cas de la régression, lorsque la capacité est trop grande, deux possibilités existent pour la réduire : augmenter le nombre d'exemples d'apprentissage ou diminuer le nombre de paramètres du système. La figure 3.1 illustre la première approche. Toutefois, dans la plupart des cas, le nombre d'exemples disponibles est limité. On doit alors se rabattre sur la deuxième option. En éliminant les variables d'entrées les moins utiles, on améliore le ratio du nombre d'exemples par rapport au nombre de paramètres à estimer et ceci nous permet de calculer un meilleur vecteur de paramètres.

Deux approches différentes ont été proposées pour implanter cette idée. La première consiste à éliminer de l'ensemble des variables d'entrées les variables qui ne sont pas significativement corrélées avec la sortie. Plusieurs algorithmes différents appliquent cette approche. Les plus utilisés sont la sélection avant, l'élimination arrière et la régression pas à pas. Ces algorithmes sont regroupés sous l'appellation générale **sélection de variables**.

L'autre approche consiste à conserver toutes les variables d'entrées, mais de pénaliser les coefficients du vecteur de paramètres en fonction de leur taille, en valeur absolue. L'objectif est de forcer les paramètres à être petits, et un petit paramètre revient à accorder peu d'importance à la variable d'entrée qui y correspond. Un paramètre ne pourra être grand que si le gain que ce paramètre permet de réaliser sur l'erreur d'apprentissage compense pour la pénalité due à sa taille. Cette idée est également à la base d'un grand nombre d'algorithmes. Dans ce mémoire, nous en présentons deux : la régression ridge et la régression ridge adaptative. L'algorithme que nous proposons est également inspiré de cette idée. L'idée de pénaliser les coefficients du vecteur de paramètres porte le nom de **régularisation** ou de **pénalisation**.

Les algorithmes qui utilisent la régularisation sont divisés en deux catégories. Ceux de la première catégorie utilisent un seul terme de pénalité regroupant tous

les paramètres. La plupart de ces algorithmes combinent l'erreur d'apprentissage à un terme de pénalité pour produire une fonction de la forme

$$C = R(\mathbf{b}, D) + \lambda P(\mathbf{b}), \tag{3.11}$$

où $P(\mathbf{b})$ est une fonction quadratique par rapport au vecteur $\mathbf{b}$. La variable λ sert à contrôler la force du terme de pénalité $P(\mathbf{b})$. Une forte pénalité forcera les paramètres à être plus petits tandis qu'une pénalité plus faible leur permettra de prendre de plus grandes valeurs. Ainsi, la taille de λ contrôle la taille de l'ensemble $\mathcal{F}$ parmi lequel on recherche la fonction optimale $\hat{f}$; plus λ est grand, et plus cet ensemble $\mathcal{F}$ sera petit. Les algorithmes qui suivent cette approche déterminent la valeur du vecteur de paramètres optimal $\hat{\mathbf{b}}$ en minimisant la fonction C. Comme des valeurs différentes de λ produiront des valeurs différentes de $\hat{\mathbf{b}}$, il faut habituellement essayer plusieurs valeurs de λ pour trouver celle qui produira le meilleur vecteur $\hat{\mathbf{b}}$. Parce que la variable λ n'est pas un des paramètres de la régression comme tel, mais plutôt un paramètre de la fonction C, et parce qu'on ne peut pas déterminer sa valeur en fonction de la minimisation de C (sinon, on prendrait toujours $\lambda = 0$), on lui donne le nom d'**hyper-paramètre**. L'algorithme de régression ridge présenté au chapitre 5 appartient à cette catégorie d'algorithmes.

Les algorithmes de la deuxième catégorie utilisent plusieurs termes de pénalités différents plutôt qu'un seul. Pour la régression linéaire, ceci permet d'associer un terme de pénalité différent à chacun des coefficients du vecteur de paramètres $\mathbf{b}$. Le vecteur optimal $\hat{\mathbf{b}}$ est déterminé en minimisant une fonction de la forme

$$C = R(\mathbf{b}, D) + \sum_m \lambda_m P(\mathbf{b}_m) \tag{3.12}$$

Associer une pénalité différente à chaque coefficient permet de forcer certains à prendre des valeurs plus proches de zéro que les autres. Ceci revient en quelque sorte à faire une sélection de variables "molle" ; c'est-à-dire qu'on n'exige pas qu'une variable soit totalement incluse ou exclue de l'ensemble d'entrée, mais on permet des niveaux "d'inclusion" plus ou moins grands pour chacune. L'avantage

de cette approche est qu'elle permet une plus grande flexibilité dans le choix des variables. La sélection de variables nous force à accepter sans retenue une variable ou à la rejeter complètement. De plus, toutes les variables acceptées sont ensuite considérées sur un pied d'égalité. Or, il peut être utile d'accorder plus d'importance à certaines des variables qu'à d'autres. Aussi, on peut parfois vouloir diminuer l'importance d'une variable sans toutefois l'éliminer totalement de l'équation de régression. L'algorithme de régression ridge adaptative du chapitre 5 détermine le vecteur de paramètres selon ce modèle. C'est également cette idée qui est à la base de l'algorithme d'optimisation des hyper-paramètres du chapitre 6.

Chapitre 4

Régression pas à pas

4.1 Introduction

Ce chapitre présente un algorithme standard en statistique pour faire de la sélection de variables (voir par exemple [9],[27],[33]). Comme mentionné à la section 3.2, il existe plusieurs algorithmes qui implantent ce concept. Parmi ceux-ci, la régression pas à pas est un de ceux qui donnent les meilleurs résultats.

L'algorithme de régression pas à pas est une procédure itérative pour choisir un sous-ensemble S des variables d'entrée. L'algorithme débute avec $S = \emptyset$. A chaque itération, on ajoute à S la variable d'entrée qui est la plus corrélée avec la sortie, en tenant compte de la corrélation de chaque variable avec celles qui sont déjà dans l'ensemble S. Par exemple, supposons qu'on ait deux variables X et Y fortement corrélées à la fois entre elles et avec la sortie, et une troisième variable Z indépendante des deux autres. Supposons également qu'à une certaine itération de l'algorithme, X fait déjà partie de l'ensemble S. Si on veut maintenant ajouter une nouvelle variable à S, laquelle choisir ? La variable Y peut paraître fortement corrélée avec la sortie, mais comme elle est aussi corrélée avec X, et que X est déjà dans S, ajouter Y à l'ensemble S nous apportera peu d'information supplémentaire. Il est donc peut-être préférable de choisir Z qui, même si elle est moins corrélée avec la sortie, apporte quand même plus d'information nouvelle que

Y n'en apporte. Une fois la nouvelle variable choisie, on vérifie si elle apporte une contribution significative à l'équation de régression. Si la contribution n'est pas significative, l'algorithme se termine. Si elle l'est, on ajoute la variable à S. Chaque variable d'entrée déjà dans S est ensuite examinée et éliminée si sa contribution n'est plus suffisamment importante. Une variable qui était la meilleure entrée à une étape précédente de l'algorithme peut maintenant être superflue à cause de sa relation avec les variables qui sont présentement dans S. L'algorithme continue ainsi jusqu'à ce qu'aucune variable n'apporte une contribution assez significative pour justifier son inclusion dans S.

Le reste du chapitre présente les détails de l'algorithme. La section 4.2 présente d'abord les hypothèses faites par l'algorithme sur la distribution des données. La section 4.3 discute ensuite comment on peut centrer les données pour simplifier les équations de ce chapitre. La section 4.4 explique comment mesurer la qualité d'une droite de régression et la section 4.5 présente une variable qui est un indice de cette qualité. Finalement, la section 4.6 regroupe les informations précédentes en un test statistique pour mesurer la contribution de chaque variable à la régression, et la section 4.7 décrit le détail de l'algorithme en utilisant ce test statistique.

4.2 Hypothèses de distribution

Plusieurs algorithmes de ce chapitre proviennent du domaine statistique. Les résultats mathématiques sur lesquels ces algorithmes sont fondées reposent sur des hypothèses sur la distribution des données. En particulier, plusieurs résultats ont été développés sous l'hypothèse que la sortie désirée est une variable aléatoire Y conditionnée par les variables d'entrée $X_1, \ldots, X_M$ selon l'équation

$$Y = \beta_0 + \beta_1 X_1 + \ldots + \beta_M X_M + \epsilon \tag{4.1}$$

où ϵ est une variable aléatoire distribuée indépendamment des X_m selon une loi normale $N(0, \sigma^2)$, et chaque valeur y^t de la variable de sortie a été générée de façon

indépendante. Nous notons ici cette hypothèse afin de pouvoir y faire référence plus loin dans le chapitre lorsque ce sera nécessaire.

4.3 Centrer les données

Plutôt que d'ajouter une colonne de uns à la matrice de données $\mathbf{X}$, il est possible d'obtenir le même résultat en soustrayant à chaque colonne de la matrice la moyenne des coefficients de cette colonne, et en faisant la même chose pour le vecteur de sortie $\mathbf{y}$. En d'autres termes, soit la matrice suivante

$$\mathbf{X}_{\text{moy}} = \begin{pmatrix} \mathbf{X}_{11} - \bar{\mathbf{X}}_1 & \mathbf{X}_{12} - \bar{\mathbf{X}}_2 & \cdots & \mathbf{X}_{1M} - \bar{\mathbf{X}}_M \\ \mathbf{X}_{21} - \bar{\mathbf{X}}_1 & \mathbf{X}_{22} - \bar{\mathbf{X}}_2 & \cdots & \mathbf{X}_{2M} - \bar{\mathbf{X}}_M \\ \vdots & \vdots & & \vdots \\ \mathbf{X}_{T1} - \bar{\mathbf{X}}_1 & \mathbf{X}_{T2} - \bar{\mathbf{X}}_2 & \cdots & \mathbf{X}_{TM} - \bar{\mathbf{X}}_M \end{pmatrix} \tag{4.2}$$

et soit

$$\mathbf{y}_{\text{moy}} = \begin{pmatrix} \mathbf{y}_1 - \bar{\mathbf{y}} \\ \mathbf{y}_2 - \bar{\mathbf{y}} \\ \vdots \\ \mathbf{y}_M - \bar{\mathbf{y}} \end{pmatrix} \tag{4.3}$$

Alors, les paramètres $\hat{\mathbf{b}}_{\text{moy}}$ obtenus en minimisant l'erreur d'apprentissage

$$R(\mathbf{b}, D_{\text{moy}}) = \frac{1}{2T}(\mathbf{X}_{\text{moy}}\mathbf{b} - \mathbf{y}_{\text{moy}})'(\mathbf{X}_{\text{moy}}\mathbf{b} - \mathbf{y}_{\text{moy}}) \tag{4.4}$$

sont les mêmes que ceux obtenus en minimisant l'équation 3.6. La seule différence est que le vecteur $\hat{\mathbf{b}}_{\text{moy}}$ a un coefficient de moins que le vecteur $\hat{\mathbf{b}}$, car il ne contient pas un terme de biais associé à aucune variable d'entrée.

On remarque également que la moyenne du vecteur $\mathbf{y}_{\text{moy}}$ est zéro. En effet

$$\bar{\mathbf{y}}_{\text{moy}} = \frac{1}{T}\sum_t (\mathbf{y}_t - \bar{\mathbf{y}}) \tag{4.5}$$

$$= \frac{1}{T}\sum_t \mathbf{y}_t - \bar{\mathbf{y}} \tag{4.6}$$

$$= \bar{\mathbf{y}} - \bar{\mathbf{y}} \tag{4.7}$$

$$= 0 \tag{4.8}$$

De même, la moyenne de chaque colonne de la matrice de donnée $\mathbf{X}_{\text{moy}}$ est également zéro. Ces deux faits seront très utiles pour simplifier la description de l'algorithme de régression pas à pas. Pour cette raison, jusqu'à la fin de ce chapitre, nous utiliserons la forme centrée de la matrice de données. Nous continuerons néanmoins à dénoter la matrice de données par $\mathbf{X}$ et le vecteur de paramètres par $\mathbf{b}$ afin d'alléger la notation.

4.4 Analyse de la variance

L'analyse de la variance est une technique pour mesurer à quel point une équation de régression explique bien les valeurs observées de la variable de sortie. La technique consiste à comparer les résultats obtenus avec le modèle

$$\mathbf{y} = \mathbf{X}\mathbf{b} \tag{4.9}$$

à ceux qu'on obtiendrait si on n'utilisait pas l'information contenue dans la matrice de données $\mathbf{X}$. Dans ce cas, notre meilleur choix serait un modèle qui prédit toujours la même valeur pour la variable de sortie. C'est-à-dire :

$$\mathbf{y} = \mathbf{1}b_0 \tag{4.10}$$

où $\mathbf{1}$ est un vecteur de la même taille que $\mathbf{y}$ et dont tous les coefficients sont des uns. Pour ce second modèle, on calcule la valeur optimale $\hat{b_0}$ du paramètre de la même façon que pour le vecteur $\hat{\mathbf{b}}$, c'est-à-dire en minimisant l'erreur empirique. L'équation de l'erreur dans ce cas est

$$R(b_0, D) = \frac{1}{2T}(\mathbf{1}b_0 - \mathbf{y})'(\mathbf{1}b_0 - \mathbf{y}) \tag{4.11}$$

Ceci peut être réécrit sous la forme

$$R(b_0, D) = \frac{1}{2T}\sum_t (b_0 - y^t)^2 \tag{4.12}$$

On voit facilement que le minimum est atteint au point

$$\hat{b_0} = \frac{1}{T}\sum_t y^t = \bar{\mathbf{y}} = 0 \tag{4.13}$$

Donc, si on n'utilise pas l'information de la matrice de données, la meilleure prédiction que l'on peut faire pour un y donné est la moyenne des valeurs observées pour la variable de sortie. Lorsque les données ont été centrées, cette moyenne est zéro.

Chacun des deux modèles explique en partie les valeurs observées de la variable de sortie. Cependant, aucun des modèles n'est parfait et il y a pour chaque valeur observée y^t une différence entre y^t et la valeur prédite $\hat{y}^t$ par le modèle. Ceci est illustré à la figure 4.1. On appelle la différence entre valeur prédite et valeur observée le **résidu**. La somme des carrées des résidus est une mesure de la variance des données autour de la droite de régression produite par le modèle. Cette somme est similaire à l'erreur d'apprentissage de chaque modèle, la seule différence étant le terme $1/2T$ qui apparaît dans l'équation de l'erreur d'apprentissage. La somme des carrés des résidus et l'erreur d'apprentissage atteignent un minimum au même point, $\hat{\mathbf{b}}$ ou $\hat{b_0}$ selon le modèle, et l'erreur d'apprentissage est simplement la moyenne des résidus, divisée par deux. Le fait de faire la moyenne ne change rien mathématiquement, mais il est plus facile de comprendre une erreur moyenne qu'une somme d'erreurs, et c'est pour cette raison que nous utilisons l'erreur d'apprentissage plutôt que la somme des carrés des résidus dans les autres chapitres de ce mémoire.

Pour simplifier leur réutilisation dans le reste de ce chapitre, nous désignons les sommes des carrés des résidus par SCR pour le premier modèle et par SYY pour le second.

$$\text{SYY} = (\mathbf{1}\hat{b_0} - \mathbf{y})'(\mathbf{1}\hat{b_0} - \mathbf{y}) \tag{4.14}$$

$$\text{SCR} = (\mathbf{X}\hat{\mathbf{b}} - \mathbf{y})'(\mathbf{X}\hat{\mathbf{b}} - \mathbf{y}) \tag{4.15}$$

Comme $\hat{b_0}$ vaut zéro, la première équation peut être simplifiée et réécrite

$$\text{SYY} = \mathbf{y}'\mathbf{y}. \tag{4.16}$$

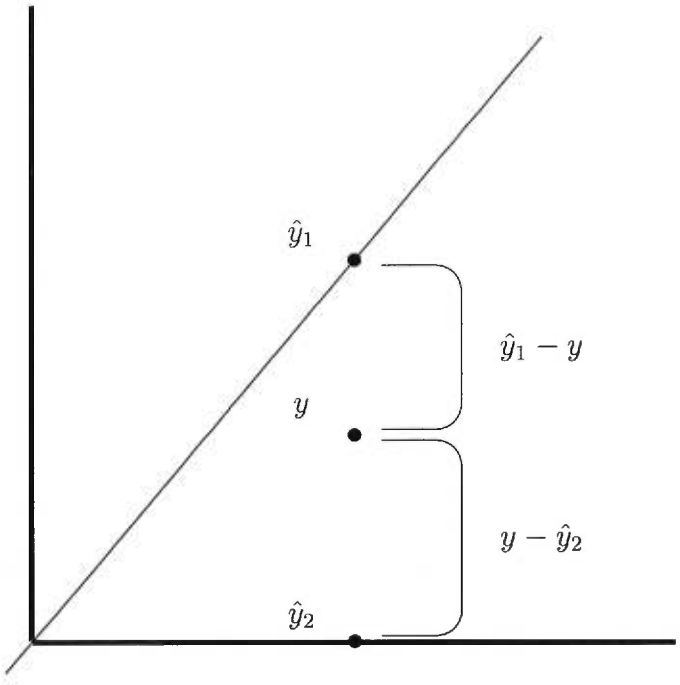


FIG. 4.1: Résidus entre les valeurs prédites par les modèles et la sortie désirée y. Pour une valeur de x donnée, le premier modèle prédit une valeur $\hat{y}_1$ qui se trouve sur la droite de régression. Le deuxième modèle prédit toujours la valeur $\hat{y}_2 = \bar{y} = 0$. La véritable sortie désirée y diffère de ces deux valeurs.

L'équation pour SCR peut également être reformulée :

$$\text{SCR} = \hat{\mathbf{b}}'\mathbf{X}'\mathbf{X}\hat{\mathbf{b}} - 2\hat{\mathbf{b}}'\mathbf{X}'\mathbf{y} + \mathbf{y}'\mathbf{y}. \tag{4.17}$$

Cette équation peut être simplifiée en utilisant l'expression pour $\hat{\mathbf{b}}$ donnée par l'équation 3.10. On obtient alors

$$\text{SCR} = \mathbf{y}'\mathbf{y} - \hat{\mathbf{b}}'\mathbf{X}'\mathbf{y}. \tag{4.18}$$

En résumé, si on n'utilise pas l'information de la matrice de données $\mathbf{X}$ et que les valeurs de la variable de sortie ont été préalablement centrées, le meilleur choix pour la régression est une droite définie par un vecteur de paramètres dont tous les coefficients sont des zéros. En deux dimensions, ceci correspondrait à l'axe des Xs. Les valeurs y observées de la variable de sortie ne sont pas réellement sur cette

droite, mais sont distribuées autour d'elle avec une certaine variance, qui peut être quantifiée par l'équation SYY. Si on utilise la matrice $\mathbf{X}$, on trouve le vecteur $\hat{\mathbf{b}}$ qui définit la droite qui explique le mieux les données. Ici encore, les valeurs observées y de la variable de sortie sont distribuées avec une certaine variance autour de cette droite, et cette variance peut être quantifiée par l'équation SCR. Cette information est résumée sous forme de tableau : la **table d'analyse de la variance**.

Source	dl	SC	MC
Régression	M	$\hat{\mathbf{b}}'\mathbf{X}'\mathbf{y}$	$\mathrm{MS}_{\mathrm{Reg}}$
Reste	$T-M-1$	$\mathbf{y}'\mathbf{y}-\hat{\mathbf{b}}'\mathbf{X}'\mathbf{y}$	s^2
Total	$T-1$	$\mathbf{y}'\mathbf{y}$	

La ligne **Total** présente la variance observée dans les données lorsqu'on n'utilise pas l'information de la matrice $\mathbf{X}$. Cette variance totale de la sortie est séparée en deux parties. La ligne **Régression** donne la quantité de la variance totale qui peut être expliquée par la meilleure hyper-droite $\hat{\mathbf{b}}$. La ligne **Reste** donne la quantité de la variance totale qui ne peut être expliquée par la régression linéaire. Chacune de ces trois lignes a un degré de liberté (dl) qui lui est associé. La somme de carrés de la ligne **Total** a un degré de liberté $T-1$, car parmi les T valeurs $y_1, \ldots, y_T$, seulement $T-1$ sont indépendantes puisque la somme des T valeurs donne zéro. Le degré de liberté de la somme de carrés de la ligne **Régression** est égal au nombre de paramètres dans le modèle. Celui de la ligne **Reste** est la différence entre les deux. La dernière colonne, **MC**, donne la moyenne des carrés de la ligne correspondante. La moyenne est faite en divisant non pas par le nombre d'observation, mais par le degré de liberté correspondant. On peut montrer que si les données obéissent à l'hypothèse de la section 4.2, alors s^2 est un estimateur non biaisé de σ^2. Ce fait est utilisé plus loin pour déterminer quelles entrées choisir pendant le processus de sélection de variables.

4.5 Coefficient de corrélation

On nomme **coefficient de corrélation** le rapport des sommes de carrés des lignes **Régression** et **Total**.

$$R^2 = \frac{\text{SYY} - \text{SCR}}{\text{SYY}} \tag{4.19}$$

Le coefficient de corrélation est une mesure de la qualité de la droite de régression trouvée. S'il est proche de un, cela signifie que la majeure partie de la variance dans les données est expliquée par la droite de régression. S'il est proche de zéro, on considérera que la droite de régression est peu utile car la variance après régression n'est pas significativement plus basse que la variance avant régression.

4.6 F-test et F-test partiel

À la section 4.1, il est dit qu'une variable n'est ajoutée à l'ensemble des entrées que si elle apporte une contribution significative à l'équation de régression. Nous présentons maintenant la méthode que nous utiliserons pour mesurer la qualité des équations de régression et ainsi déterminer si une variable doit être sélectionnée comme entrée ou non.

On peut montrer que si l'hypothèse de la section 4.2 est vérifiée, alors lorsque tous les coefficients $\beta_1, \ldots, \beta_M$ sont zéro, le ratio

$$F = \frac{\text{MS}_{\text{Reg}}}{s^2} \tag{4.20}$$

est distribué selon une loi F avec M et $T - M - 1$ degrés de liberté. On peut donc utiliser ceci pour tester si les coefficients $\beta_1 = \ldots = \beta_M = 0$. Le **F-test** consiste à comparer la valeur calculée de F avec le point $100(1 - \alpha)\%$ de la distribution F(M,$T - M - 1$) pour déterminer si les coefficients peuvent être considérés comme valant zéro en se basant sur les données observées. Si F est plus grand que la valeur du point choisi de la distribution, l'hypothèse comme

quoi les coefficients sont zéro est rejetée et l'équation de régression obtenue est considérée significative.

Le principe du F-test peut être généralisé de la façon suivante. Supposons que nous appliquons deux fois la régression linéaire avec un nombre différent de variables d'entrées à chaque fois, et toujours la même variable de sortie. Supposons que la première fois, on utilise seulement p des M variables d'entrées disponibles, tandis que la deuxième fois, on utilise q variables, avec q plus grand que p. Les q variables de la deuxième régression sont les p variables de la première, auxquelles on en ajoute $q - p$ nouvelles. On obtient ainsi deux vecteurs de paramètres.

$$\hat{\mathbf{b}}_1 = (\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_p) \tag{4.21}$$

$$\hat{\mathbf{b}}_2 = (\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_p, \hat{b}_{p+1}, \ldots, \hat{b}_q) \tag{4.22}$$

Pour chacun des deux vecteurs, on a une somme des carrées des résidus, SCR_1 et SCR_2, ainsi qu'un estimé de la variance, s_1^2 et s_2^2, qui lui correspont. Supposons maintenant que la variable de sortie a été générée selon le modèle

$$Y = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p + \beta_{p+1} X_{p+1} + \ldots + \beta_q X_q + \epsilon \tag{4.23}$$

et que ce modèle obéit aux hypothèses de la section 4.2. On peut montrer que si les coefficients $\beta_{p+1} = \ldots = \beta_q = 0$, alors le ratio

$$F_{\text{partiel}} = \frac{\mathrm{SCR}_1 - \mathrm{SCR}_2}{s_2^2} \tag{4.24}$$

est distribué selon une loi $\mathrm{F}(q - p,\, T - q - 1)$. Comme pour le F-test, on peut utiliser ce rapport pour déterminer si les coefficients $\beta_{p+1}, \ldots, \beta_q$ sont différents de zéro ou, en d'autres termes, si l'ajout des $q-p$ variables additionnelles améliore significativement l'équation de régression. Si le ratio F_{partiel} est plus grand que le point $100(1 - \alpha)\%$ de la distribution, alors l'hypothèse que les coefficients sont zéro peut être rejetée et donc l'ajout des variables d'entrée additionnelles est significatif.

Le **F-test partiel** est un autre cas particulier de la procédure du paragraphe précédent. Un F-test partiel est utilisé pour tester si le fait d'ajouter une nouvelle

variable à une équation de régression améliore celle-ci de façon significative. Le F-test partiel obéit donc à la procédure ci-haut, mais avec q fixé à $p+1$. C'est ce test qui est utilisé par la régression pas à pas pour choisir quelle variable ajouter à l'ensemble des variables d'entrée, ainsi que pour tester à chaque itération si une variable déjà dans l'ensemble devrait être éliminée ou non.

4.7 Algorithme détaillé

Nous donnons maintenant la description complète et détaillée de l'algorithme de régression pas à pas. Dans ce qui suit, nous dénotons par U l'ensemble de toutes les variables d'entrée et par S l'ensemble des entrées sélectionnées pour être utilisées dans la régression. À chaque itération de l'algorithme, nous utiliserons aussi l'ensemble temporaire S' pour désigner l'ensemble S auquel est ajouté la variable d'entrée présentement sous considération. Voici donc l'algorithme.

(i) Initialiser $S \leftarrow \emptyset$.

(ii) Soit X la variable d'entrée la plus corrélée avec la sortie. Poser $S' \leftarrow \{X\}$, calculer le vecteur de paramètres $\hat{\mathbf{b}}$ correspondant et utiliser le F-test pour déterminer si la régression est significative. Si elle ne l'est pas, l'algorithme s'arrête et l'ensemble d'entrées S est l'ensemble vide. Si la régression est significative, poser $S \leftarrow S'$ et passer à l'étape suivante.

(iii) Parmi toutes les variables d'entrées qui appartiennent à $U - S$, soit X celle dont le F-test partiel produit la plus grande valeur de F_{partiel}. Si cette valeur de F_{partiel} n'est pas assez grande pour que la variable soit jugée significative selon le critère énoncé à la section 4.6, alors l'algorithme s'arrête. Si la variable est significative, alors poser $S' \leftarrow S \cup \{X\}$, puis pour chaque variable dans S', calculer la valeur de F_{partiel} que l'on aurait obtenu si cette variable avait été la dernière à être sélectionnée pour faire partie de S. Soit X' la variable d'entrée dont la valeur de F_{partiel} est la plus petite. Si X' est significative, alors poser $S \leftarrow S'$. Sinon, poser $S \leftarrow S' - \{X'\}$.

(iv) Répeter l'étape 3 jusqu'à ce qu'il n'y ait plus de variable pour laquelle le F-test partiel justifie son inclusion dans S, ou jusqu'à ce que $S = U$.

Dans notre implantation, tous les F-tests et F-tests partiels ont été fait à un niveau α de 5%.

Chapitre 5

Régression ridge adaptative

Cette section présente un algorithme de régression linéaire basé sur l'idée de régularisation. Comme expliqué à la section 3.2, la régularisation consiste à ajouter un terme de pénalité à l'erreur d'apprentissage et de choisir comme vecteur de paramètres le vecteur qui minimise la combinaison de ces deux termes. Comme c'était le cas pour la sélection de variables, il existe plusieurs algorithmes qui utilisent la régularisation. Nous en présentons un relativement récent : la régression ridge adaptative. Cet algorithme est basé sur un autre algorithme, plus ancien mais très couramment utilisé, la régression ridge.

5.1 Régression ridge

5.1.1 Algorithme

L'algorithme de régression ridge a été initialement proposé par Hoerl en 1962 [13] et est discuté en détail par Hoerl et Kennard [15]. L'idée est d'ajouter à l'erreur d'apprentissage un terme pénalisant également tous les coefficients du vecteur de paramètres et dont la taille est contrôlée par un hyper-paramètre λ.

$$C(\lambda, \mathbf{b}, D) = \frac{1}{2}(\mathbf{Xb} - \mathbf{y})'(\mathbf{Xb} - \mathbf{y}) + \lambda \sum_m b_m^2 \tag{5.1}$$

En dérivant cette équation par rapport aux coefficients b_m et en solutionnant au point où les dérivées valent zéro, on obtient la valeur de $\mathbf{b}$ qui minimise l'équation.

$$\hat{\mathbf{b}}_r = (\mathbf{X}'\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}'\mathbf{y} \tag{5.2}$$

A cause du terme de pénalité, $\hat{\mathbf{b}}_r$ produit une plus grande erreur d'apprentissage que le vecteur $\hat{\mathbf{b}}$ produit par la régression linéaire classique, mais pour plusieurs valeurs de λ, sur de nouveaux exemples—i.e. des exemples qui ne sont pas dans l'ensemble d'apprentissage—il produira habituellement une plus petite erreur que $\hat{\mathbf{b}}$.

5.1.2 Interprétation géométrique

On peut voir l'algorithme de cette section comme une régression linéaire où les paramètres sont sujets à une restriction de la forme

$$\mathbf{b}'\mathbf{b} \leq c^2. \tag{5.3}$$

Nous sommes alors en présence d'un problème d'optimisation sous contrainte, que l'on peut solutionner en utilisant la méthode des multiplicateurs de Lagrange [1]. Soit

$$L = (\mathbf{X}\mathbf{b} - \mathbf{y})'(\mathbf{X}\mathbf{b} - \mathbf{y}) + \lambda(\mathbf{b}'\mathbf{b} - c^2), \tag{5.4}$$

la méthode de Lagrange nous dit qu'il existe un vecteur $\mathbf{b}$ qui minimise l'erreur d'apprentissage sous la contrainte 5.3, si et seulement si il existe un λ tel que

$$\frac{\partial L}{\partial \mathbf{b}} = 0, \tag{5.5}$$

$$\mathbf{b}'\mathbf{b} \leq c^2, \tag{5.6}$$

$$\lambda \geq 0, \tag{5.7}$$

$$\lambda(\mathbf{b}'\mathbf{b} - c^2) = 0. \tag{5.8}$$

Remarquez que la solution de l'équation 5.5 est donnée par l'équation 5.2. La seule différence est que λ est maintenant relié à la borne c^2 par l'entremise des contraintes 5.6, 5.7 et 5.8.

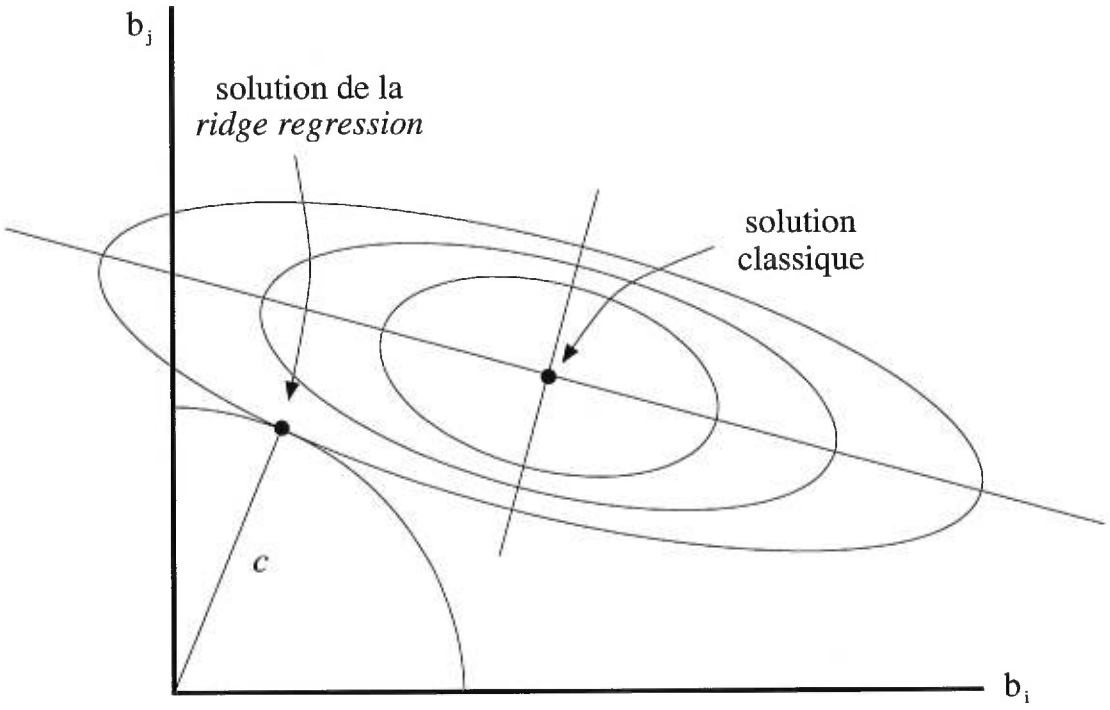


FIG. 5.1: Solution produite par la régression linéaire classique et solution produite par l'algorithme de régression ridge. La solution classique correspond au résultat obtenu par la régression ridge adaptative avec un hyper-paramètre $\lambda = 0$.

Deux cas sont possibles pour satisfaire ces trois contraintes : ou bien $\lambda = 0$ et $\mathbf{b}'\mathbf{b} < c^2$, ou bien $\lambda > 0$ et $\mathbf{b}'\mathbf{b} = c^2$. Le premier de ces deux cas nous ramène à la solution classique du problème de régression ; ce cas survient lorsque la borne c^2 est trop grande pour jouer un rôle dans la détermination de $\hat{\mathbf{b}}_r$. Le deuxième cas survient lorsque la borne c^2 pousse la solution vers zéro. La borne c^2 imposée sur la solution produit un hyper-paramètre $\lambda > 0$ par l'entremise des trois contraintes, et cet hyper-paramètre produit à son tour un vecteur $\hat{\mathbf{b}}_r$ qui correspond à la solution optimale du problème de régression sous contrainte. La situation est illustrée à la figure 5.1.

La figure présente un exemple en deux dimensions, c'est-à-dire pour deux coefficients du vecteur de paramètres. La valeur du vecteur $\hat{\mathbf{b}}$ qui minimise l'erreur d'apprentissage est indiquée comme étant la solution classique. Autour de ce

point sont indiqués les "lignes de niveau", où la valeur de l'erreur d'apprentissage est constante. Comme l'erreur d'apprentissage est une fonction quadratique des paramètres, ces lignes de niveaux sont des ellipses. La contrainte de l'équation 5.3 est un cercle de rayon c centré à l'origine. Le vecteur $\hat{\mathbf{b}}_r$ produit par l'algorithme de régression ridge se trouve à l'intersection de la contrainte et de la ligne de niveau la plus près possible du minimum.

Cette analyse montre que pour un λ donné, l'algorithme recherche la fonction optimale $\hat{f}$ parmi un ensemble $\mathcal{F}_\lambda$ qui est une fonction de ce λ. Cet ensemble $\mathcal{F}_\lambda$ est le cercle de rayon c centré à l'origine. On peut montrer que plus la valeur de λ est petite, plus grand sera le rayon. De plus, pour tout λ et λ', si $\lambda < \lambda'$, alors $\mathcal{F}_{\lambda'} \subset \mathcal{F}_\lambda$. En effet, comme $\mathcal{F}_{\lambda'}$ et $\mathcal{F}_\lambda$ sont des cercles centrées à l'origine, le plus petit doit nécessairement être inclus dans le plus grand. Comme la capacité de l'algorithme est proportionnelle à la taille de l'ensemble $\mathcal{F}_\lambda$, ceci implique que plus λ est petit, plus la capacité sera grande. Or, on sait que l'erreur de généralisation produite par tout algorithme est une fonction de sa capacité. Selon la figure 2.5, il existe une capacité optimale qui produira la plus petite erreur de généralisation. Comme la capacité est dans ce cas-ci contrôlée par λ, il doit donc exister une valeur optimale de λ qui produira la plus petite erreur de généralisation.

5.1.3 Deuxième interprétation géométrique

Le problème peut être examiné sous un autre angle. Considérons la forme générale de la fonction de coût minimisée par l'algorithme.

$$C(\lambda, \mathbf{b}, D) = E(\mathbf{b}, D) + \lambda P(\mathbf{b}); \tag{5.9}$$

La fonction à minimiser est un terme d'erreur $E(\mathbf{b}, D)$ auquel on ajoute une pénalité $P(\mathbf{b})$ qui est fonction des paramètres. Le terme d'erreur est une fonction quadratique et a la forme

$$E(\mathbf{b}, D) = E_0 + \boldsymbol{\gamma}'\mathbf{b} + \frac{1}{2}\mathbf{b}'\mathbf{H}\mathbf{b} \tag{5.10}$$

où le scalaire E_0, le vecteur $\boldsymbol{\gamma}$ et la matrice hessienne $\mathbf{H}$ sont constants par rapport à $\mathbf{b}$. Pour la régression ridge, la pénalité est

$$P(\mathbf{b}) = \frac{1}{2}\mathbf{b}'\mathbf{b}. \tag{5.11}$$

La régression linéaire classique produit le vecteur $\hat{\mathbf{b}}$ en minimisant le terme d'erreur donné par l'équation 5.10. Le minimum est atteint au point où les dérivés par rapport aux coefficients b_m valent zéro. Le vecteur $\hat{\mathbf{b}}$ satisfait donc

$$\boldsymbol{\gamma} + \mathbf{H}\hat{\mathbf{b}} = 0. \tag{5.12}$$

La régression ridge quant à elle produit le vecteur $\hat{\mathbf{b}}_r$ en minimisant la combinaison du terme d'erreur et de la pénalité. Ce minimum est également atteint au point où les dérivés valent zéro. Le vecteur $\hat{\mathbf{b}}_r$ satisfait donc

$$\boldsymbol{\gamma} + \mathbf{H}\hat{\mathbf{b}}_r + \lambda\mathbf{1}'\hat{\mathbf{b}}_r = 0. \tag{5.13}$$

On peut maintenant visualiser l'effet du terme de pénalité si on effectue une rotation des axes de l'espace des paramètres pour diagonaliser la matrice $\mathbf{H}$. Comme $\mathbf{H}$ est une matrice hessienne, elle est symétrique et il est donc possible de trouver ses M vecteurs propres orthonormaux $\mathbf{u}_m$. Ces vecteurs satisfont l'équation

$$\mathbf{H}\mathbf{u}_m = \omega_m\mathbf{u}_m \tag{5.14}$$

où les ω_m sont les valeurs propres de la matrice $\mathbf{H}$. Les M vecteurs propres forment une base de l'espace des paramètres et il est donc possible d'exprimer $\hat{\mathbf{b}}$ et $\hat{\mathbf{b}}_r$ comme une combinaison linéaire de ces vecteurs.

$$\hat{\mathbf{b}} = \sum_m [\hat{b}_m]_{\mathbf{u}}\mathbf{u}_m \tag{5.15}$$

$$\hat{\mathbf{b}}_r = \sum_m [\hat{b}_{r,m}]_{\mathbf{u}}\mathbf{u}_m \tag{5.16}$$

Ici, nous avons utilisé la notation $[\hat{b}_m]_{\mathbf{u}}$ pour dénoter le m-ième coefficient du vecteur $\hat{\mathbf{b}}$ dans la base définie par les vecteurs propres $\mathbf{u}$. En combinant les cinq

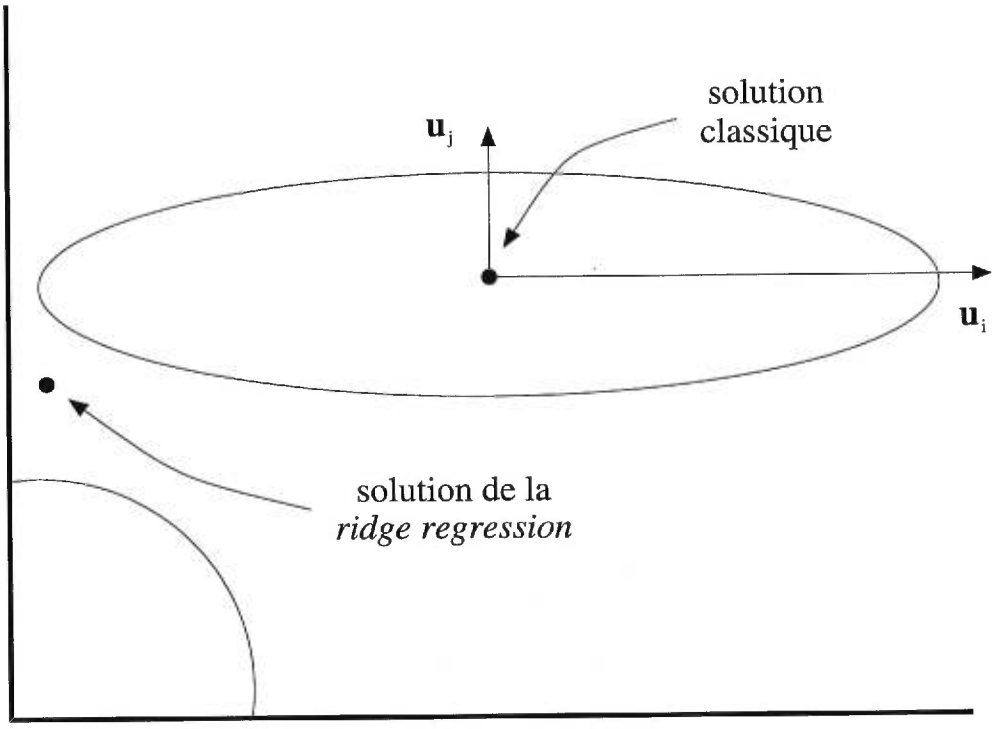


FIG. 5.2: Deuxième interprétation géométrique de la solution produite par la régression linéaire classique et de la solution produite par l'algorithme de régression ridge. L'ellipse représente une courbe de niveau de l'erreur $E(b, D)$. Le cercle centré à l'origine est une courbe de niveau le long de laquelle la pénalité $P(b)$ est constante.

dernières équations, et en utilisant le fait que les vecteurs propres sont orthonormaux, on obtient la relation suivante entre les vecteurs $\hat{\mathbf{b}}$ et $\hat{\mathbf{b}}_r$:

$$[\hat{b}_{r,m}]_{\mathbf{u}} = \frac{\omega_m}{\omega_m + \lambda}[\hat{b}_m]_{\mathbf{u}} \tag{5.17}$$

La situation est illustrée à la figure 5.2. Les vecteurs propres $\mathbf{u}_m$ sont orientés le long des axes principaux des courbes de niveaux de la fonction d'erreur $E(\mathbf{b}, D)$. Une petite valeur propre signifie $\omega_m \ll \lambda$ et donc $[\hat{b}_{r,m}]_{\mathbf{u}} \ll [\hat{b}_m]_{\mathbf{u}}$. Le vecteur $\hat{\mathbf{b}}_r$ est donc "poussé vers zéro" le long de cette direction. Lorsque le vecteur propre est court, alors $\omega_m \gg \lambda$ et $[\hat{b}_{r,m}]_{\mathbf{u}} \approx [\hat{b}_m]_{\mathbf{u}}$. Le vecteur $\hat{\mathbf{b}}_r$ est donc proche du vecteur $\hat{\mathbf{b}}$ dans cette direction.

5.1.4 Interprétation Bayesienne

On peut également interpréter la fonction de coût de la régression ridge comme la combinaison de l'erreur sur les données et d'une distribution a priori des coefficients b_m. Cette approche Bayesienne au problème de régression s'applique non seulement à la régression linéaire mais aussi à la régression non-linéaire. Elle a été récemment particulièrement étudiée par Mackay [19], qui a entre autres montré comment elle peut être appliquée aux réseaux de neurones multicouches pour obtenir une distribution sur les poids d'un réseau.

Minimisation de l'erreur vue comme maximum de vraisemblance

La première chose à constater est que la minimisation de l'erreur $E(\mathbf{b}, D)$ peut être considéré comme une maximisation de la vraisemblance pour la distribution des données décrite à la section 4.2. Considérons la distribution jointe des entrées et des sorties désirées : pour un seul exemple d'apprentissage, on a la fonction de probabilité

$$p(\mathbf{x}, y) = p(y|\mathbf{x})p(\mathbf{x}). \tag{5.18}$$

L'objectif de la régression est de trouver les paramètres du modèle qui maximisent la vraisemblance de l'ensemble d'apprentissage. On dénote cette vraisemblance $\mathcal{L}$. En supposant que tous les exemples sont tirés indépendamment de la même distribution, cette vraisemblance s'écrit

$$\mathcal{L} = \prod_t p(\mathbf{x}^t, y^t) \tag{5.19}$$

$$= \prod_t p(y^t|\mathbf{x}^t)p(\mathbf{x}^t). \tag{5.20}$$

Plutôt que de maximiser la vraisemblance, il est plus facile de minimiser

$$E = -\ln \mathcal{L} = -\sum_t \ln p(y^t|\mathbf{x}^t) - \sum_t \ln p(\mathbf{x}^t). \tag{5.21}$$

On dénote cette fonction par la lettre E car on peut la considérer comme une erreur à minimiser. Comme on le verra, cette fonction est aussi la fonction $E(\mathbf{b}, D)$

de la section précédente, d'où l'utilisation de la même lettre pour les désigner. Le deuxième terme de l'équation 5.21 est indépendant des paramètres de la régression. On peut donc l'éliminer sans affecter la solution. La nouvelle forme de E est alors

$$E = -\ln \mathcal{L} = -\sum_t \ln p(y^t|\mathbf{x}^t). \tag{5.22}$$

Maintenant, en supposant que les données sont distribuées selon le modèle de la section 4.2, on a la relation suivante entre la sortie prédite $\mathbf{b}'\mathbf{x}$ et la sortie désirée y :

$$y = \mathbf{b}'\mathbf{x} + \epsilon. \tag{5.23}$$

où ϵ est un bruit Gaussien $N(0, \sigma^2)$. La distribution de ϵ est

$$p(\epsilon) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left(-\frac{\epsilon^2}{2\sigma^2}\right) \tag{5.24}$$

En combinant les équations 5.23 et 5.24, on obtient l'expression suivante pour la distribution de la sortie désirée :

$$p(y|\mathbf{x}) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left(-\frac{(\mathbf{b}'\mathbf{x} - y)^2}{2\sigma^2}\right) \tag{5.25}$$

Si on insère ce résultat dans l'équation 5.22, on obtient

$$E = \frac{1}{2\sigma^2}\sum_t (\mathbf{b}'\mathbf{x}^t - y^t)^2 + C \tag{5.26}$$

où C est une constante indépendante du vecteur $\mathbf{b}$. Comme notre but est de minimiser E par rapport à $\mathbf{b}$, cette constante peut donc être éliminée. De même, le terme σ^2 n'a aucune influence sur le point où le minimum est atteint. En enlevant ces deux quantités, on retrouve l'expression

$$E(\mathbf{b}, D) = (\mathbf{X}\mathbf{b} - \mathbf{y})'(\mathbf{X}\mathbf{b} - \mathbf{y}) \tag{5.27}$$

qui est l'équation de l'erreur quadratique que nous avons utilisé dans les sections précédentes.

Application du théorème de Bayes

Nous avons montré comment la minimisation de l'erreur quadratique peut être vue comme la maximisation de la vraisemblace des données lorsque celles-ci sont issues d'une distribution normale. L'approche classique en régression linéaire choisit le vecteur de paramètres optimal en maximisant cette vraisemblance. L'approche Bayesienne quant à elle utilise une méthode différente.

L'idée de l'approche Bayesienne est de considérer que les paramètres sont distribués selon une certaine fonction de densité. En l'absence de données d'apprentissage, on suppose que les paramètres sont distribués selon une certaine densité a priori $p(\mathbf{b})$. En présence d'un ensemble d'apprentissage, on utilise le théorème de Bayes pour calculer la nouvelle distribution des paramètres $p(\mathbf{b}|D)$. Le vecteur de paramètres le plus probable est alors celui qui maximise cette nouvelle distribution des paramètres. Ceci est illustré à la figure 5.3. On peut montrer que pour une certaine densité a priori des paramètres (une normale de moyenne zéro), le vecteur optimal après l'application du théorème de Bayes est le vecteur $\hat{\mathbf{b}}_r$ produit par la régression ridge adaptative.

Le théorème de Bayes, lorsque appliqué à la distribution des paramètres et des données, produit l'équation

$$p(\mathbf{b}|D) = \frac{p(D|\mathbf{b})p(\mathbf{b})}{p(D)}. \tag{5.28}$$

Le dénominateur $p(D)$ est un terme de normalisation défini par

$$p(D) = \int p(D|\mathbf{b})p(\mathbf{b})d\mathbf{b} \tag{5.29}$$

et assure que l'intégrale de $p(\mathbf{b}|D)$ lorsque faite sur tout l'espace des paramètres $\mathbf{b}$ donne un. Le dénominateur est donc indépendant de $\mathbf{b}$ et pourra être éliminé lorsqu'on cherchera à maximiser $p(\mathbf{b}|D)$. La fonction $p(D|\mathbf{b})$ est la vraisemblance

$$\mathcal{L} = p(D|\mathbf{b}) = \prod_t p(y^t|\mathbf{x}^t). \tag{5.30}$$

En y substituant l'équation 5.25 à la place de $p(y^t|\mathbf{x}^t)$, on obtient

$$p(D|\mathbf{b}) = C_1 \exp\left(-\frac{\beta}{2}\sum_t (\mathbf{b}'\mathbf{x}^t - y^t)^2\right) \tag{5.31}$$

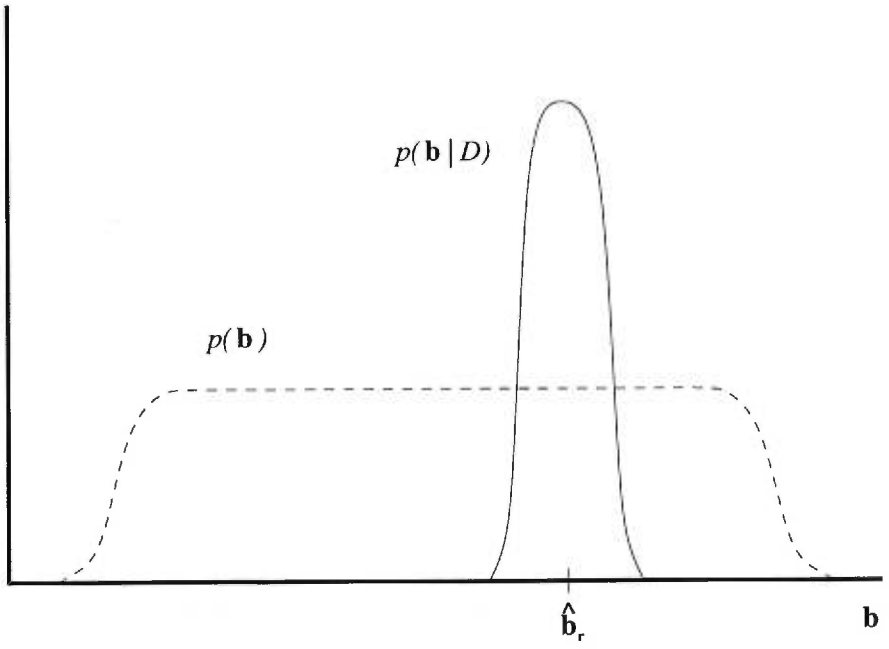


FIG. 5.3: Fonction de densité des paramètres a priori et a posteriori. La courbe pointillée représente la densité a priori et la courbe pleine, la densité a posteriori. Le vecteur $\hat{\mathbf{b}}_r$ produit par l'algorithme de régression ridge est celui qui maximise la probabilité a posteriori.

où C_1 et β sont deux constantes indépendantes de $\mathbf{b}$.

Il reste à choisir une fonction pour la disribution a priori $p(\mathbf{b})$. La fonction qui produira la fonction de coût de la régression ridge est

$$p(\mathbf{b}) = C_2 \exp\left(-\frac{\alpha}{2}\mathbf{b}'\mathbf{b}\right) \tag{5.32}$$

où C_2 et α sont deux constantes indépendantes de $\mathbf{b}$, tout comme C_1 et β. La fonction 5.32 est une Gaussienne centrée à l'origine et dont la variance est $1/\alpha$. Les courbes de niveaux de la fonction sont les mêmes que celles illustrées à la figure 5.1. Il s'agit donc d'une pénalité sur les paramètres identique à celle des sections précédentes.

On peut maintenant combiner les équations 5.28, 5.31 et 5.32. On obtient

$$p(\mathbf{b}|D) = \frac{C}{p(D)} \exp\left(-\frac{\beta}{2}\sum_t (\mathbf{x}'\mathbf{b} - y)^2 - \frac{\alpha}{2}\mathbf{b}'\mathbf{b}\right). \tag{5.33}$$

avec $C = C_1 \times C_2$. Comme notre but est de maximiser cette équation par rapport au vecteur $\mathbf{b}$, la constante $C/p(D)$ peut être éliminée. Maximiser l'équation est alors équivalent à minimiser

$$-\ln p(\mathbf{b}|D) = \frac{\beta}{2}\sum_t(\mathbf{b}'\mathbf{x}^t - y^t)^2 + \frac{\alpha}{2}\mathbf{b}'\mathbf{b}. \tag{5.34}$$

Remarquez qu'ici, les valeurs précises de α et β importent peu ; ce qui compte est plutôt le ratio α/β. On peut donc fixer β à deux et remplacer α par $2 \times \lambda$ et on retrouve alors l'équation de coût 5.1

Conclusion

Le vecteur de paramètres $\hat{\mathbf{b}}_r$ produit par la régression ridge est le vecteur qu'on obtient lorsqu'on cherche à maximiser la vraisemblance des paramètres en fonction des données d'apprentissage et qu'on ajoute une distribution a priori sur les paramètres qui favorise les petites valeurs des coefficients. Remarquez comment l'expression $p(D|\mathbf{b})$ dans l'équation 5.28 est le terme que l'approche classique cherche à maximiser. L'expression $p(\mathbf{b}|D)$ maximisée par la régression ridge est la multiplication de cette densité avec la densité a priori sur les paramètres $p(\mathbf{b})$, normalisée pour que l'intégrale donne un—cette normalisation étant nécessaire pour que $p(\mathbf{b}|D)$ soit bien une densité.

5.2 *Non-negative garotte* et lasso

Plusieurs algorithmes de régression linéaire se sont inspirés de la régression ridge. La régression ridge adaptative par exemple en est un. Deux autres sont le *non-negative garotte* et le lasso. Nous discutons brièvement de ces deux algorithmes avant de présenter la régression ridge adaptative.

Le *non-negative garotte* a été proposé en 1993 par Léo Breiman [6]. Cet algorithme produit le vecteur de paramètres en minimisant

$$\sum_t(\sum_m c_m \hat{b}_m x_m^t - y^t)^2 \tag{5.35}$$

sur c_m et $\hat{b}_m$, sous les contraintes

$$c_m \geq 0 \tag{5.36}$$

$$\sum_m c_m \leq C \tag{5.37}$$

où C est un hyper-paramètre qui doit être choisi à l'avance, et les coefficients $\hat{b}_m$ sont les coefficients du vecteur de paramètres $\hat{\mathbf{b}}$ produit par la régression linéaire classique. Une fois C choisi, les paramètres c_m sont sélectionnés de façon à minimiser l'équation 5.35 tout en satisfaisant les contraintes.

S'inspirant de la *garotte*, Robert Tibshirani a proposé un algorithme similaire mais qui ne dépend pas du vecteur de paramètres $\hat{\mathbf{b}}$. Son algorithme, le lasso [29], détermine la valeur du vecteur de paramètre en minimisant

$$\sum_t (\sum_m b_m x_m^t - y^t)^2 \tag{5.38}$$

sous la contrainte

$$\sum_m |b_m| \leq C \tag{5.39}$$

où C est ici encore un hyper-paramètre qui doit être sélectionné à l'avance. Le calcul du vecteur de paramètres optimal est un problème d'optimisation quadratique sujet à des contraintes linéaires. La solution est calculable mais complexe à implanter.

Le lasso a eu une influence importante dans le développement de la régression ridge adaptative. En particulier, il a été démontré [10] que les deux algorithmes produisent des vecteurs de paramètres identiques lorsque leurs ensembles d'apprentissage sont les mêmes. La régression ridge adaptative a toutefois l'avantage d'être beaucoup plus simple à implanter que le lasso. Pour cette raison, nous avons préféré cet algorithme pour nos expériences.

5.3 Régression ridge adaptative

La régression ridge adaptative est une extension de l'algorithme de régression ridge qui a été récemment proposée par Yves Grandvalet [10]. L'idée derrière

l'algorithme est de pénaliser chaque coefficient b_m du vecteur de paramètres séparément et de combiner les M termes de pénalité ainsi obtenus en utilisant M hyper-paramètres différents. Le grand nombre d'hyper-paramètres que cette approche ajoute au problème constitue toutefois un problème potentiel. Il est en effet difficile de trouver la valeur optimale des hyper-paramètres. Essayer un grand nombre de combinaisons de valeurs possibles pour les M hyper-paramètres est difficile. Il faut donc un algorithme qui permettra de faire une recherche dans l'espace des hyper-paramètres. La régression ridge adaptative propose un tel algorithme qui est à la fois efficace et facile à implanter.

La régression ridge adaptative détermine la valeur du vecteur de paramètres en minimisant

$$C(\lambda, \mathbf{b}, D) = \frac{1}{2}(\mathbf{Xb} - \mathbf{y})'(\mathbf{Xb} - \mathbf{y}) + \sum_m \lambda_m b_m^2 \tag{5.40}$$

sous les contraintes

$$\frac{1}{M}\sum_m \frac{1}{\lambda_m} = \frac{1}{\mu} \tag{5.41}$$

$$\lambda_m > 0 \tag{5.42}$$

Ici, il n'y a véritablement qu'un seul hyper-paramètre, μ, qui doit être sélectionné à l'avance. Une fois μ choisi, les λ_m peuvent être optimisés en même temps que les coefficients b_m.

L'algorithme d'optimisation est le suivant. Tout d'abord, on définit deux vecteurs $\boldsymbol{\gamma}$ et $\mathbf{c}$ avec les coefficients

$$\gamma_m = \sqrt{\frac{\lambda_m}{\mu}} b_m \tag{5.43}$$

$$c_m = \sqrt{\frac{\mu}{\lambda_m}} \tag{5.44}$$

Le processus d'optimisation consiste à calculer successivement

$$c_m^{(s)^2} = \frac{M\gamma_m^{(s-1)^2}}{\sum_m \gamma_m^{(s-1)^2}} \tag{5.45}$$

$$\gamma^{(s)} = (\mathrm{diag}(\mathbf{c}^{(s)})\mathbf{X}'\mathbf{X}\mathrm{diag}(\mathbf{c}^{(s)}) + \mu\mathbf{I})^{-1}\mathrm{diag}(\mathbf{c}^{(s)})\mathbf{X}'\mathbf{y} \tag{5.46}$$

où diag($\mathbf{c}$) est une matrice carrée dont la diagonale est le vecteur $\mathbf{c}$ et dont les autres coefficients sont zéro. L'algorithme peut être initialisé en utilisant ou bien les paramètres produits par la régression ridge, ou bien ceux produits par la régression linéaire classique. L'hyper-paramètre μ doit être choisi par essai et erreur, en utilisant un estimateur de l'erreur de généralisation pour évaluer la qualité de chaque valeur de μ. Cet estimateur peut être soit une borne théorique, soit un estimateur calculé avec des données qui n'ont pas servies à l'apprentissage, comme la validation croisée.

5.3.1 Dérivation de l'algorithme d'optimisation

L'approche directe proposée par Boukari et Grandvalet [5] pour minimiser l'équation 5.40 sous les contraintes 5.41 et 5.42 est la suivante. Tout d'abord, calculer le vecteur $\mathbf{b}$ selon la même méthode que celle utilisée pour solutionner l'équation 5.1. Ceci donne l'expression suivante pour $\mathbf{b}$:

$$\mathbf{b} = (\mathbf{X}'\mathbf{X} + \mathbf{\Lambda})^{-1}\mathbf{X}'\mathbf{y} \tag{5.47}$$

où la matrice diagonale $\mathbf{\Lambda} = \text{diag}(\lambda_1, \ldots, \lambda_m)$. Ensuite, utiliser le vecteur $\mathbf{b}$ pour recalculer de nouveaux coefficients λ_m :

$$\lambda_m = \mu \frac{\sum_j |b_j|}{M|b_m|}. \tag{5.48}$$

L'idée étant de pénaliser plus fortement les petits coefficients b_m en leur assignant un λ_m plus élevé, tout en respectant les contrainte 5.41 et 5.42. On répète ainsi la mise à jour de $\mathbf{b}$ et $\mathbf{\Lambda}$ jusqu'à ce que les coefficients b_m et λ_m convergent vers des valeurs stables.

En pratique, il a été découvert que cette approche tend à pousser certains des λ_m vers l'infini. Cette difficulté a pu être solutionnée en introduisant la paramétrisation décrite par les équations 5.43 et 5.44. Sous cette nouvelle paramétrisation, le problème consiste maintenant à minimiser

$$C(\mathbf{b}, \mathbf{c}, \boldsymbol{\gamma}) = \sum_t (\sum_m \gamma_m c_m x_m^t - y^t)^2 + \mu \sum_m \gamma_m^2 \tag{5.49}$$

sous les contraintes

$$\sum_m c_m^2 = M \tag{5.50}$$

$$c_m \geq 0 \tag{5.51}$$

La minimisation de l'équation 5.49 doit ici être faite à la fois par rapport à $\mathbf{c}$ et à $\boldsymbol{\gamma}$. Si on introduit la nouvelle variable

$$\mathbf{Z} = \text{diag}(\mathbf{c})\mathbf{X} \tag{5.52}$$

alors l'équation 5.49 peut être réécrite sous forme matricielle

$$C(\mathbf{b}, \mathbf{c}, \boldsymbol{\gamma}) = (\mathbf{Z}\boldsymbol{\gamma} - \mathbf{y})'(\mathbf{Z}\boldsymbol{\gamma} - \mathbf{y}) + \mu \sum_m \gamma_m^2 \tag{5.53}$$

Cette équation a la même forme que l'équation 5.1 et la même technique peut être utilisée pour la minimiser par rapport à $\boldsymbol{\gamma}$. En réappliquant la même approche que celle proposée pour minimiser l'équation 5.40, on obtient l'équation suivante pour $\boldsymbol{\gamma}$

$$\boldsymbol{\gamma} = (\mathbf{Z}'\mathbf{Z} + \mu\mathbf{I})^{-1}\mathbf{Z}'\mathbf{y} \tag{5.54}$$

et cette autre équation pour $\mathbf{c}$

$$c_m^{(s)^2} = \frac{M\gamma_m^{(s-1)^2}}{\sum_m \gamma_m^{(s-1)^2}} \tag{5.55}$$

En substituant l'équation 5.52 dans 5.54, on obtient les équations 5.45 et 5.46. On évalue successivement ces deux équations jusqu'à convergence vers des valeurs stables [11].

5.3.2 Interprétation géométrique

L'interprétation géométrique de la régression ridge adaptative est similaire à celle de la régression ridge. Si on compare les équations 5.1 et 5.40, on remarque que la seule différence est que la régression ridge utilise un seul paramètre λ pour pénaliser tous les coefficients, tandis que la régression ridge adaptative utilise M paramètres λ_m pour pénaliser chaque coefficient séparément. Ceci a pour effet que

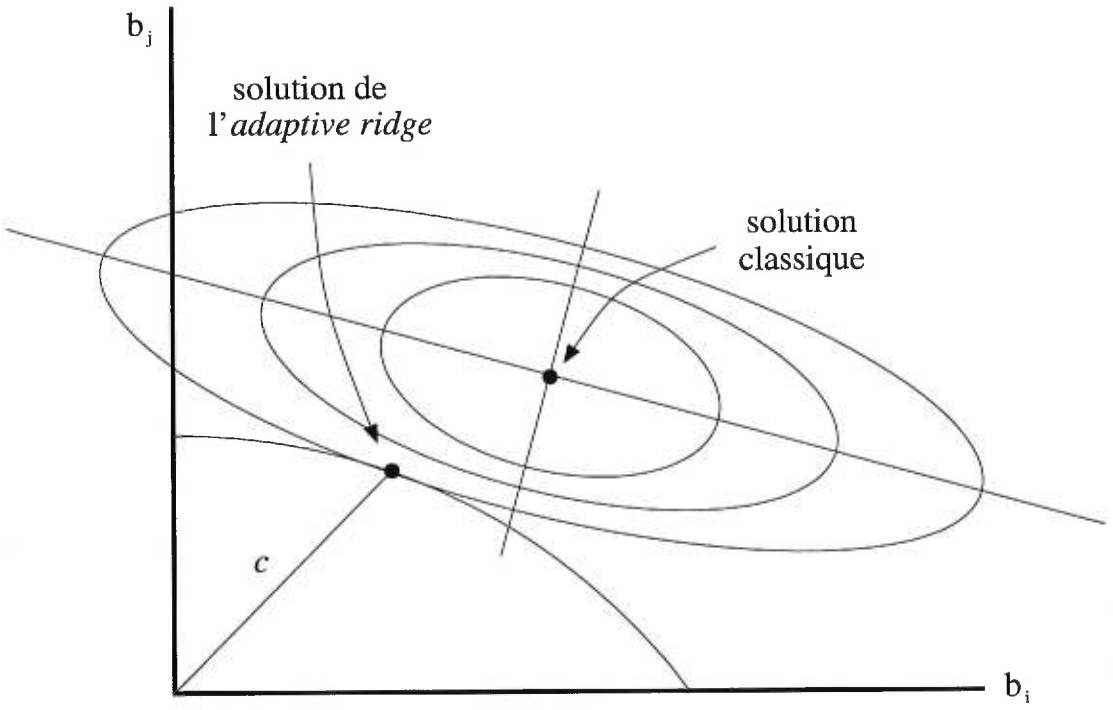


FIG. 5.4: Solution produite par la régression linéaire classique et solution produite par l'algorithme de régression ridge adaptative.

la contrainte sur les paramètres n'est plus sphérique, mais a plutôt la forme d'une ellipse : on permet à certains coefficients de s'éloigner beaucoup de zéro, tandis qu'on en force d'autres à demeurer très près de l'origine. La situation est illustrée à la figure 5.4, qui devrait être comparée à la figure 5.1. Plus un paramètre λ_m est grand, plus le rayon de l'ellipse dans cette direction sera petit. La contrainte 5.41 restreint le volume de l'ellipse : plus l'hyper-paramètre μ est grand, plus le volume total sera petit et plus les paramètres b_m seront poussés vers zéro.

5.3.3 Interprétation Bayesienne

L'interprétaion Bayesienne de l'algorithme est également similaire à celle de la régression ridge. La différence ici est la forme de la distribution a priori $p(\mathbf{b})$. Plutôt que d'avoir une distribution unique pour tous les coefficients b_m, on a

plutôt une distribution différente pour chaque coefficient.

$$p(b_m) = C_m \exp\left(-\frac{\alpha_m}{2} b_m^2\right) \tag{5.56}$$

La variance de chaque distribution est proportionnelle à $1/\alpha_m$. Ces distributions sont combinées pour produire la distribution a priori du vecteur $\mathbf{b}$.

$$p(\mathbf{b}) = C' \exp\left(-\sum_m \frac{\alpha_m}{2} b_m^2\right) \tag{5.57}$$

où $C' = C_1 \times \ldots \times C_m$. Cette distribution, lorsque combinée à $p(D|\mathbf{b})$ produit l'équation suivante pour $p(\mathbf{b}|D)$:

$$p(\mathbf{b}|D) = \frac{C}{p(D)} \exp\left(-\frac{\beta}{2}\sum_t (\mathbf{x}'\mathbf{b} - y)^2 - \sum_m \frac{\alpha_m}{2} b_m^2\right). \tag{5.58}$$

Maximiser la vraisemblance revient à minimiser

$$-\ln p(\mathbf{b}|D) = \frac{\beta}{2}\sum_t (\mathbf{x}'\mathbf{b} - y)^2 + \sum_m \frac{\alpha_m}{2} b_m^2. \tag{5.59}$$

En divisant par $\beta/2$ et en posant $\lambda_m = \alpha_m/\beta$, on obtient l'équation 5.40. La variance a priori de chaque coefficient étant proportionnelle à $1/\alpha_m$, elle est donc inversement proportionnelle au λ_m correspondant. Plus un coefficient λ_m est grand, plus on restreint les valeurs que le paramètre b_m peut prendre. La contrainte 5.42 fait en sorte que la moyenne des variances des paramètres b_m est proportionnelle $1/\mu$.

5.3.4 Détails d'implantation

Pour mesurer la qualité de chaque valeur de l'hyper-paramètre μ nous avons utilisé la validation croisée pour calculer l'estimateur de l'erreur de généralisation. La valeur de K dans la validation croisée a été fixée à 10. Pour sélectionner la valeur optimale de μ, nous avons utilisé la méthode de Powell décrite à la page 406 du livre *Numerical Recipes* [28]. Pour le processus d'optimisation des vecteurs $\mathbf{c}$ et $\boldsymbol{\gamma}$, le nombre maximum d'itérations était de dix. Si la convergence n'était pas encore atteinte à ce moment, les valeurs calculées à la dixième itération étaient utilisées. Cette limite a été determinée après quelques tentatives et apparaît comme un bon compromis entre temps de calcul et précision de la solution.

Chapitre 6

Optimisation des hyper-paramètres

6.1 Introduction

Les quatres algorithmes présentés à la section précédente ont plusieurs points en commun :

- Chaque algorithme utilise un seul hyper-paramètre.
- Pour une valeur donnée de l'hyper-paramètre, l'algorithme produit toujours le même vecteur de paramètres. Le choix de ce vecteur est donc une fonction de l'hyper-paramètre.
- La valeur optimale de l'hyper-paramètre doit être déterminée par essai et erreur. Le choix de l'hyper-paramètre ne fait pas partie de l'algorithme.

Ces trois caractéristiques sont typiques d'un grand nombre d'algorithmes d'apprentissage, non seulement en régression linéaire mais dans tous les champs de la théorie. Par exemple, la régularisation est souvent utilisée pour amélirorer la performance des réseaux de neurones. La technique porte alors le nom de *weight decay* [12] et un hyper-paramètre est utilisé pour contrôler la force du terme de pénalité.

La plupart des algorithmes utilisent un seul hyper-paramètre parce que les hyper-paramètres sont difficiles à optimiser. Comme il n'existe pas d'algorithme pour trouver la valeur des hyper-paramètres qui produira le meilleur vecteur de paramètres, il n'y a pas d'autre solution que d'essayer un grand nombre de valeurs différentes et de choisir celle qui donne le meilleur résultat. Ceci est acceptable quand le nombre d'hyper-paramètres est petit, un ou deux par exemple, mais le nombre de valeurs possibles pour les hyper-paramètres croît exponentiellement quand le nombre d'hyper-paramètres augmente. S'il n'y a qu'un seul hyper-paramètre, tester N valeurs n'est pas trop difficile. Mais si l'algorithme compte d hyper-paramètres, alors tester N valeurs pour chacun exige de tester N^d combinaisons de valeurs différentes. Ce type de problème, où une méthode est acceptable en basse dimension mais devient rapidement trop complexe quand le nombre de dimensions augmente, survient fréquemment dans le domaine des algorithmes d'apprentissage. Il survient assez souvent pour que les chercheurs lui donnent un nom : c'est la **malédiction de la dimensionalité**.

L'algorithme de cette section utilise un vecteur d'hyper-paramètres qui est optimisé en utilisant une méthode à base de gradient. Avoir plusieurs hyper-paramètres permet de faire de la sélection de variables "molle" en assignant un terme de pénalité différent à chaque variable d'entrée. Chaque entrée est ainsi pénalisée indépendamment des autres. La taille de la pénalité est inversement proportionnelle à "l'utilité" de cette entrée. Contrairement à la véritable sélection de variables qui exige que chaque entrée soit ou bien dans l'ensemble des entrées choisies, ou bien hors de l'ensemble, cet algorithme permet un "degré de sélection" différent pour chaque variable d'entrée. L'algorithme est décrit à la section 6.4. Mais d'abord, nous introduisons une nouvelle terminologie pour décrire le problème de sélection des hyper-paramètres.

6.2 Terminologie

Prenons l'ensemble d'apprentissage D et séparons-le en deux sous-ensembles distincts D_{ent} et D_{val}. Nous nommerons D_{ent} l'ensemble d'entraînement et D_{val} l'ensemble de validation.

Les algorithmes d'apprentissage décrits dans ce mémoire calculent tous la valeur du vecteur de paramètre en minimisant une fonction de coût. Nous nommerons cette fonction le **critère d'entraînement** et nous la désignerons par la lettre C. Le critère d'entraînement est une fonction des paramètres, des hyper-paramètres et de l'ensemble d'entraînement. Le vecteur de paramètres sélectionné par l'algorithme est celui qui minimise ce critère :

$$\mathbf{b}^* = \mathbf{b}^*(\boldsymbol{\lambda}, D_{\text{ent}}) = \text{argmin}_{\mathbf{b}} C(\boldsymbol{\lambda}, \mathbf{b}, D_{\text{ent}}) \tag{6.1}$$

où $\boldsymbol{\lambda}$ est le vecteur d'hyper-paramètres. S'il y a un seul hyper-paramètre, alors $\boldsymbol{\lambda}$ est un vecteur de taille unitaire.

Une fois que le vecteur de paramètres est choisi, sa qualité est mesurée en utilisant un estimateur de l'erreur de généralisation. Cet estimateur est calculé sur des exemples qui n'ont pas servis à l'entraînement, d'où notre division de l'ensemble d'apprentissage en deux parties distinctes. Nous nommerons cet estimateur le **critère de sélection de modèle** et nous le désignerons par la lettre E. Le critère de sélection de modèle est une fonction de l'ensemble de validation et du vecteur de paramètres $\mathbf{b}^*$. Mais comme $\mathbf{b}^* = \mathbf{b}^*(\boldsymbol{\lambda}, D_{\text{ent}})$ est en fait une fonction du vecteur d'hyper-paramètres et de l'ensemble d'entraînement, le critère de sélection de modèle est vraiment une fonction du vecteur d'hyper-paramètres $\boldsymbol{\lambda}$ et de l'ensemble d'apprentissage D : $E = E(\boldsymbol{\lambda}, D)$. Et tout comme il existe un $\mathbf{b}^*$ qui minimise $C(\boldsymbol{\lambda}, \mathbf{b}, D_{\text{ent}})$, il existe un $\boldsymbol{\lambda}^*$ qui minimise $E(\boldsymbol{\lambda}, D)$.

$$\boldsymbol{\lambda}^* = \boldsymbol{\lambda}^*(D) = \text{argmin}_{\boldsymbol{\lambda}} E(\boldsymbol{\lambda}, D) \tag{6.2}$$

Si $\boldsymbol{\lambda}$ possède un seul coefficient, la minimisation peut se faire par essai et erreur. Si sa taille est plus grande, ce n'est pas si facile. Mais si le critère de sélection de

modèle est différentiable par rapport aux hyper-paramètres, on peut alors utiliser une méthode à base de gradient. Le diagramme 6.1 résume la situation.

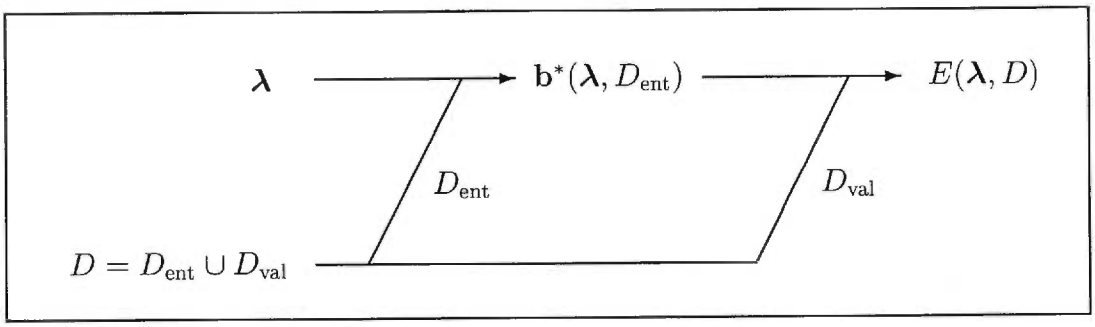


FIG. 6.1: Calcul de l'estimateur de l'erreur de généralisation E en fonction des hyper-paramètres $\boldsymbol{\lambda}$ et de l'ensemble d'apprentissage D

Notez que la distinction entre ensemble d'entraînement et ensemble de validation n'est pas nécessairement toujours aussi simple. Le critère de sélection de modèle peut être n'importe quelle fonction qui est un estimateur de l'erreur de généralisation, comme par exemple la validation croisée.

6.3 Exemples

Le critère d'entraînement de l'algorithme de régression pas à pas est l'erreur d'apprentissage décrite au chapitre 2.

$$C_{\text{rpe}}(\mathbf{b}, \boldsymbol{\lambda}, D_{\text{ent}}) = \frac{1}{|D_{\text{ent}}|} \sum_{\mathbf{z}^t \in D_{\text{ent}}} Q(\mathbf{b}, \mathbf{z}^t) \tag{6.3}$$

Cet algorithme n'utilise pas d'hyper-paramètre et n'a donc pas besoin de critère de sélection de modèle. La division de l'ensemble d'apprentissage est donc $D = D_{\text{ent}}$ et $D_{\text{val}} = \emptyset$. Il n'y a pas de fonction $E(\boldsymbol{\lambda}, D_{\text{val}})$.

L'algorithme de régression ridge adaptative utilise un seul hyper-paramètre : $\boldsymbol{\lambda} = (\mu)$. Son critère d'entraînement est l'erreur d'apprentissage régularisée par

un terme de pénalité.

$$C_{\text{ar}}(\boldsymbol{\lambda}, \mathbf{b}, D_{\text{ent}}) = \frac{1}{|D_{\text{ent}}|} \sum_{\mathbf{z}^t \in D_{\text{ent}}} Q(\mathbf{b}, \mathbf{z}^t) + \sum_m \lambda_m b_m^2 \tag{6.4}$$

où les paramètres λ_m sont calculés en fonction de l'hyper-paramètre μ selon les contraintes 5.41 et 5.42. Le critère de sélection de modèle pourrait être n'importe quel estimateur de l'erreur de généralisation. Dans les expériences présentées dans ce mémoire, nous utilisons la validation croisée.

$$E_{\text{ar}}(\boldsymbol{\lambda}, D) = \frac{1}{K} \sum_i \frac{1}{|D_{\text{val}}^i|} \sum_{\mathbf{z}^t \in D_{\text{val}}^i} Q(\mathbf{b}^*(\boldsymbol{\lambda}, D_{\text{ent}}^i), \mathbf{z}^t) \tag{6.5}$$

Notez que la formulation des algorithmes en terme de critères d'entraînement et de sélection de modèle n'est pas standard dans la littérature. Les concepts eux-même sont bien connus, ainsi par exemple la description de la régression ridge adaptative dit bien de choisir la valeur de l'hyper-paramètre en utilisant un estimateur de l'erreur de généralisation, mais la terminologie utilisée est la nôtre. Elle est utile parce qu'elle met en relief le fait que les algorithmes minimisent deux fonctions de coût, une pour le vecteur de paramètre et l'autre pour les hyper-hyperparamètres, et rend aussi apparent le lien entre les hyper-paramètres et l'estimateur de l'erreur de généralisation, en particulier la possibilié d'utiliser le gradient pour minimiser l'estimateur en fonction des hyper-paramètres.

6.4 Optimisation simple des hyper-paramètres

Nous proposons une nouvelle combinaison de critère d'entraînement et critère de sélection de modèle. Comme mentionné plus haut, la combinaison que nous proposons a ceci de particulier que, contrairement aux algorithmes proposés jusqu'ici, elle utilise plusieurs hyper-paramètres et il est possible de calculer la dérivée du critère de sélection de modèle par rapport aux hyper-paramètres.

Le critère d'entraînement que nous proposons est le suivant :

$$C(b, \boldsymbol{\lambda}, D_{\text{ent}}) = \frac{1}{|D_{\text{ent}}|} \sum_{\mathbf{z}_t \in D_{\text{ent}}} Q(\mathbf{b}, \mathbf{z}_t) + \sum_m \lambda_m^2 b_m^2. \tag{6.6}$$

C'est essentiellement le même critère que celui utilisé par la régression ridge adaptative. Cependant, plutôt que d'avoir une contrainte exigeant que les λ_m soient positifs, on utilise le carré des hyper-paramètres dans le critère. On abandonne également la contrainte 5.42 sur la somme des inverses des λ_m. Donc, contrairement à la régression ridge adaptative, où il y avait un seul hyper-paramètre μ et où les λ_m pouvaient être optimisés par rapport au critère d'entraînement, ici les λ_m deviennent de véritables hyper-paramètres et ne sont sujets à aucune contrainte : ils seront optimisés selon le critère de sélection de modèle.

Plusieurs fonctions différentes peuvent être utilisés comme critère de sélection de modèle. Nous utiliserons le même que pour la régression ridge adaptative : la validation croisée.

$$E(\lambda, D) = \frac{1}{K} \sum_i \frac{1}{|D^i_{\text{val}}|} \sum_{\mathbf{z}^t \in D^i_{\text{val}}} Q(\mathbf{b}^*(\boldsymbol{\lambda}, D^i_{\text{ent}}), \mathbf{z}^t) \tag{6.7}$$

Une fois ces deux équations choisies, il nous faut un algorithme pour calculer le vecteur d'hyper-paramètres $\boldsymbol{\lambda}^*$ et le vecteur de paramètres $\mathbf{b}^*$ qui minimisent ces deux équations. La minimisation du critère d'entraînement peut se faire analytiquement et est discutée à la section 6.4.1. Cette section discute comment calculer $\mathbf{b}^*$ pour un $\boldsymbol{\lambda}$ fixe. La minimisation du critère de sélection de modèle quant à elle n'a pas de solution analytique. Pour trouver $\boldsymbol{\lambda}^*$, on utilise plutôt une méthode à base de gradient. La section 6.4.2 explique comment calculer le gradient de E par rapport à $\boldsymbol{\lambda}$. Pour effectuer cette opération, il faut pouvoir propager le gradient à travers le calcul de $\mathbf{b}^*$. Ceci est le sujet de la section 6.4.3.

6.4.1 Calcul des paramètres

La méthode que nous utilisons pour calculer le vecteur de paramètres $\mathbf{b}^*$ est valide non seulement pour le critère d'entraînement proposé, mais aussi pour toute fonction $C(\boldsymbol{\lambda}, \mathbf{b}, D_{\text{ent}})$ qui est quadratique par rapport aux paramètres $\mathbf{b}$, et qui est continue et différentiable presque partout [4]. Dans ce qui suit, nous décrivons

la procédure pour le cas général d'un critère quadratique, tout en indiquant le cas particulier qu'est le critère d'entraînement de l'équation 6.6. Notez que la méthode peut aussi être appliquée dans le cas d'un critère d'entraînement non quadratique. Le détail de la procédure pour ce cas est donné dans [2].

Dans le cas général, un critère d'entraînement quadratique par rapport aux paramètres $\mathbf{b}$ a la forme

$$C(\boldsymbol{\lambda}, \mathbf{b}, D_{\text{ent}}) = w_0 + \mathbf{w}'\mathbf{b} + \frac{1}{2}\mathbf{b}'\mathbf{H}\mathbf{b} \tag{6.8}$$

où $w_0 = w_0(\boldsymbol{\lambda}, D_{\text{ent}})$ est un scalaire, $\mathbf{w} = \mathbf{w}(\boldsymbol{\lambda}, D_{\text{ent}})$ est un vecteur de la même taille que le vecteur de paramètres $\mathbf{b}$, c'est-à-dire de taille M, et $\mathbf{H} = \mathbf{H}(\boldsymbol{\lambda}, D_{\text{ent}})$ est une matrice $M \times M$ qui est définie positive. Ces trois variables sont des fonctions des hyper-paramètres et de l'ensemble d'entraînement. Le minimum de l'équation se trouve au point $\mathbf{b}^*$ où le gradient vaut zéro.

$$\left.\frac{\partial C}{\partial \mathbf{b}}\right|_{\mathbf{b}^*} = \mathbf{w} + \mathbf{H}\mathbf{b}^* = 0 \tag{6.9}$$

La solution de cette équation nous donne une expression pour le vecteur de paramètres $\mathbf{b}^*$ de l'équation 6.1.

$$\mathbf{b}^* = \mathbf{b}^*(\lambda, D_{\text{ent}}) = -\mathbf{H}^{-1}\mathbf{w} \tag{6.10}$$

Nous avons donc maintenant un algorithme pour calculer un $\mathbf{b}^*$ pour un vecteur d'hyper-paramètres $\boldsymbol{\lambda}$ donné dans le cas où le critère d'entraînement est quadratique par rapport au vecteur de paramètres. Comment appliquer ceci au critère que nous avons proposé ? Reformulons notre critère d'entraînement pour qu'il s'applique non seulement au cas où la sortie désirée est un scalaire y, mais aussi pour le cas où elle est un vecteur $\mathbf{y}$ de taille arbitraire N. Dans ce cas, la régression se fait alors indépendamment pour chacune des N sorties désirées. On peut imaginer qu'on effectue N régressions différentes, pour calculer N vecteurs de paramètres $\mathbf{b}^n$ différents. Seulement, chaque hyper-paramètre λ_m est partagé par toutes les régressions, c'est-à-dire qu'un même hyper-paramètre λ_m pénalise

les coefficients b_m^n pour toutes les valeurs de n. Le critère d'entraînement devient

$$C(\boldsymbol{\lambda}, \mathbf{B}, D_{\text{ent}}) = \frac{1}{|D_{\text{ent}}|} \sum_{\mathbf{z}^t \in D_{\text{ent}}} \vec{Q}(\mathbf{B}, z^t) + \sum_m \lambda_m \sum_n B_{n,m}^2 \tag{6.11}$$

où $\mathbf{B}$ est une matrice $N \times M$ dont chaque ligne est un des N vecteurs $\mathbf{b}^n$ de taille M. La fonction $\vec{Q}$ est une généralisation de la fonction Q pour le cas vectoriel :

$$\vec{Q}(\mathbf{B}, \mathbf{z}) = \frac{1}{2}(\mathbf{Bx} - \mathbf{y})'(\mathbf{Bx} - \mathbf{y}). \tag{6.12}$$

L'équation 6.11 a la même forme que l'équation 6.8. Pour les rendre égales, il suffit de poser

$$w_0 = \frac{1}{2} \sum_t \|\mathbf{y}^t\|^2 \tag{6.13}$$

$$\mathbf{w}_{(i \times j)} = -\sum_t y_i^t x_j^t \tag{6.14}$$

$$\mathbf{H}_{(i \times j),(i' \times j')} = \delta_{i,i'} \sum_t x_j^t x_{j'}^t + \delta_{i,i'} \delta_{j,j'} \lambda_j \tag{6.15}$$

Grâce à ces trois équations, on peut appliquer au critère proposé tous les algorithmes qui fonctionnent dans le cas général d'un critère quadratique.

6.4.2 Gradient du critère de sélection

Comme mentionné plus haut, le vecteur d'hyper-paramètres optimal $\boldsymbol{\lambda}^*$ ne peut être calculé analytiquement. Il est toutefois possible de calculer le gradient du critère de sélection de modèle par rapport aux hyper-paramètres et d'utiliser ensuite une méthode à base de gradient pour trouver le minimum de la fonction. Le gradient de $E(\boldsymbol{\lambda}, D)$ par rapport à un hyper-paramètre λ_m est donné par l'équation

$$\frac{dE}{d\lambda_m} = \sum_{m'} \frac{dE}{db_{m'}^*} \frac{db_{m'}^*}{d\lambda_m}. \tag{6.16}$$

La première des deux dérivés est facile à calculer :

$$\frac{dE}{db_{m'}^*} = \frac{1}{K} \sum_i \frac{1}{|D_{\text{val}}^i|} \sum_{\mathbf{z}^t \in D_{\text{val}}^i} \frac{dQ(\mathbf{b}^*, \mathbf{z}^t)}{db_{m'}^*}. \tag{6.17}$$

Le gradient de $Q(\mathbf{b}^*, \mathbf{z})$ par rapport à $b^*_{m'}$ est aussi très simple :

$$\frac{dQ(\mathbf{b}^*, \mathbf{z})}{db^*_{m'}} = (\mathbf{x}'\mathbf{b}^* - y)x_{m'}. \tag{6.18}$$

La deuxième dérivé dans la somme de l'équation 6.16 peut être calculée à partir de l'équation 6.10. La matrice $\mathbf{H}$ et le vecteur $\mathbf{w}$ dans cette équation sont tous deux des fonctions du vecteur d'hyper-paramètres $\boldsymbol{\lambda}$. La dérivée de cette équation par rapport à λ_m donne

$$\frac{db^*_{m'}}{d\lambda_m} = -\sum_j \frac{dH^{-1}_{m',j}}{d\lambda_m} w_j - \sum_j H^{-1}_{m',j} \frac{dw_j}{d\lambda_m}. \tag{6.19}$$

Cette équation exige de calculer le gradient à travers une inversion de matrice. Bien que ceci soit possible (voir [2]), le temps de calcul requis est important : le nombre d'opérations d'additions et de multiplications exigées est de l'ordre de $2M^3$. Une meilleure approche est d'utiliser directement l'équation 6.9. Ceci est le sujet de la section suivante.

6.4.3 La décomposition de Cholesky

Calculer le gradient à travers l'inversion d'une matrice est coûteux. La question se pose alors : est-il possible d'éviter d'avoir à propager le gradient à travers l'inversion ? La réponse est oui. Revenons à l'équation 6.9. Cette équation nous donne une relation entre $\mathbf{b}^*$ et $\boldsymbol{\lambda}$ et, indirectement, nous donne donc aussi une relation entre $\frac{dE}{d\mathbf{b}^*}$ et $\frac{dE}{d\boldsymbol{\lambda}}$. Avec cette équation, il est possible d'utiliser une méthode numérique pour calculer $\mathbf{b}^*$ à partir de $\boldsymbol{\lambda}$ sans avoir à inverser la matrice $\mathbf{H}$ et en effectuant seulement $M^3/3$ additions et multiplications. Nous expliquons plus loin la technique pour y arriver. De plus, chacune de ces opérations est différentiable. Il est donc possible de *rétro-propager* le gradient à travers chaque opération, et ainsi obtenir $\frac{dE}{d\boldsymbol{\lambda}}$ à partir de $\frac{dE}{d\mathbf{b}^*}$. Chaque opération qui a été effectuée pour calculer $\mathbf{b}^*$ à partir de $\boldsymbol{\lambda}$ est "inversée" pour calculer $\frac{dE}{d\boldsymbol{\lambda}}$ à partir de $\frac{dE}{d\mathbf{b}^*}$. Et ceci demande seulement le même nombre d'opérations qu'il n'en faut pour calculer $\mathbf{b}^*$, soit environ $M^3/3$.

Pour calculer $\mathbf{b}^*$ à partir de $\boldsymbol{\lambda}$ avec l'équation 6.9, nous utilisons une technique de décomposition de matrice portant le nom de **décomposition de Cholesky**. Soit $\mathbf{H}$ une matrice définie positive et symétrique. Il existe une matrice $\mathbf{L}$ triangulaire inférieure telle que

$$\mathbf{H} = \mathbf{L}\mathbf{L}'. \tag{6.20}$$

On peut écrire l'équation précécente sous la forme de $M \times M$ équations impliquant les coefficients L_{ij}. La solution de ce système d'équations nous donne les deux équations suivantes pour les coefficients de $\mathbf{L}$:

$$L_{ii} = \left(H_{ii} - \sum_{k=1}^{i-1} L_{ik}^2\right)^{1/2} \tag{6.21}$$

$$L_{ji} = \frac{1}{L_{ii}}\left(H_{ij} - \sum_{k=1}^{i-1} L_{ik}L_{jk}\right) \qquad j = i+1, i+2, \ldots, M \tag{6.22}$$

Si on applique ces deux équations dans l'ordre $i = 1, 2, \ldots, M$, on remarque que les coefficents de $\mathbf{L}$ qui apparaissent du côté droit sont déjà déterminés au moment où en a besoin.

Une fois la matrice $\mathbf{L}$ obtenue, on peut réécrire l'équation 6.9 sous la forme

$$\mathbf{L}\mathbf{L}'\mathbf{b} = -\mathbf{w}. \tag{6.23}$$

On solutionne cette équation en trouvant d'abord un vecteur $\mathbf{u}$ tel que

$$\mathbf{L}\mathbf{u} = -\mathbf{w} \tag{6.24}$$

et on résoud ensuite

$$\mathbf{L}'\mathbf{b} = \mathbf{u}. \tag{6.25}$$

Parce que $\mathbf{L}$ est triangulaire inférieure, la solution de l'équation 6.24 est facile à calculer.

$$u_1 = -\frac{w_1}{L_{11}} \tag{6.26}$$

$$u_i = -\frac{1}{L_{ii}}\left[w_i + \sum_{j=1}^{i-1} L_{ij}u_j\right] \qquad i = 2, 3, \ldots, M \tag{6.27}$$

De même pour l'équation 6.25. Cette fois, on profite du fait que $\mathbf{L}'$ est triangulaire supérieure.

$$b_M^* = \frac{u_M}{L_{MM}} \tag{6.28}$$

$$b_i^* = \frac{1}{L_{ii}}\left[u_i - \sum_{j=i+1}^{M} L_{ji} b_j\right] \qquad i = M-1, M-2, \ldots, 1 \tag{6.29}$$

Le calcul du gradient $\frac{dE}{d\boldsymbol{\lambda}}$ à partir de $\frac{dE}{d\mathbf{b}^*}$ se fait par les mêmes opérations mais dans l'ordre inverse. On propage le gradient d'abord à travers la solution de $\mathbf{L}'\mathbf{b} = \mathbf{u}$ en utilisant les équations 6.28 et 6.29. Cette étape calcule le gradient par rapport à $\mathbf{L}$ et $\mathbf{u}$, à partir du gradient par rapport à $\mathbf{b}^*$.

initialiser dEdb $\leftarrow \frac{dE}{d\mathbf{b}^*}$
initialiser dEdL $\leftarrow \vec{0}$
initialiser dEdu $\leftarrow \vec{0}$
pour $i = 1, \ldots, M$
    $\text{dEdu}_i \leftarrow \text{dEdb}_i / L_{i,i}$
    $\text{dEdL}_{i,i} \leftarrow \text{dEdL}_{i,i} - \text{dEdb}_i \, b_i / L_{i,i}$
    for $k = i+1 \ldots s$
        $\text{dEdb}_k \leftarrow \text{dEdb}_k - \text{dEdb}_i \, L_{k,i} / L_{i,i}$
        $\text{dEdL}_{k,i} \leftarrow \text{dEdL}_{k,i} - \text{dEdb}_i \, b_k / L_{i,i}$

Ensuite, on propage à travers la solution de $\mathbf{Lu} = -\mathbf{w}$ en utilisant les équations 6.26 et 6.27. Cette étape calcule le gradient par rapport à $\mathbf{w}$ et modifie le gradient par rapport à $\mathbf{L}$.

pour $i = M, \ldots, 1$
    $\frac{dE}{dw_i} \leftarrow -\text{dEdu}_i / L_{i,i}$
    $\text{dEdL}_{i,i} \leftarrow \text{dEdL}_{i,i} - \text{dEdu}_i \, u_i / L_{i,i}$
    pour $k = 1, \ldots, i-1$
        $\text{dEdu}_k \leftarrow \text{dEdu}_k - \text{dEdu}_i \, L_{i,k} / L_{i,i}$
        $\text{dEdL}_{i,k} \leftarrow \text{dEdL}_{i,k} - \text{dEdu}_i \, u_k / L_{i,i}$

La prochaine étape est de calculer le gradient par rapport à $\mathbf{H}$ en propageant le gradient par rapport à $\mathbf{L}$ à travers la décomposition de Cholesky.

pour $i = s, \ldots, 1$

  pour $j = s, \ldots, i+1$

    $\mathrm{dEdL}_{i,i} \leftarrow \mathrm{dEdL}_{i,i} - \mathrm{dEdL}_{j,i} L_{j,i} / L_{i,i}$

    $\frac{dE}{dH_{i,j}} \leftarrow \mathrm{dEdL}_{j,i} / L_{i,i}$

    pour $k = 1, \ldots, i-1$

      $\mathrm{dEdL}_{i,k} \leftarrow \mathrm{dEdL}_{i,k} - \mathrm{dEdL}_{j,i} L_{j,k} / L_{i,i}$

      $\mathrm{dEdL}_{j,k} \leftarrow \mathrm{dEdL}_{j,k} - \mathrm{dEdL}_{j,i} L_{i,k} / L_{i,i}$

  $\frac{dE}{dH_{i,i}} \leftarrow \frac{1}{2} \mathrm{dEdL}_{i,i} / L_{i,i}$

  pour $k = 1, \ldots, i-1$

    $\mathrm{dEdL}_{i,k} \leftarrow \mathrm{dEdL}_{i,k} - \mathrm{dEdL}_{i,i} L_{i,k} / L_{i,i}$

Remarquez que comme $\mathbf{H}$ est symétrique, nous n'avons qu'à calculer les éléments qui font partie du triangle supérieur de la matrice. Après ces trois étapes, nous disposons des gradients de E par rapport à $\mathbf{H}$ et $\mathbf{w}$. Comme ces deux vecteurs sont des fonctions de $\boldsymbol{\lambda}$, on peut calculer le gradient de E par rapport aux coefficients λ_m de la façon suivante :

$$\frac{dE}{d\lambda_m} = \sum_i \frac{\partial E}{\partial w_i} \frac{dw_i}{d\lambda_m} + \sum_{i,j} \frac{\partial E}{\partial H_{i,j}} \frac{dH_{i,j}}{d\lambda_m}. \tag{6.30}$$

6.5 Détails d'implantation

Certains détails d'implantation n'ont pas été fournis dans la description de l'algorithme. Nous n'avons entre autres pas discuté le détail du processus d'optimisation. Nous avons donné une méthode pour calculer le gradient $\frac{dE}{d\boldsymbol{\lambda}}$, mais nous n'avons pas spécifié comment utiliser ce gradient. La raison pour laquelle nous ne l'avons pas fait est qu'il existe plusieurs algorithmes différents pour calculer le minimum d'une fonction en utilisant le gradient. Ces algorithmes ont pour nom descente de gradient, gradients conjugués ou encore descente de gradient stochas-

tique, pour ne nommer que ceux-là. Tous ces algorithmes utilisent le fait que le gradient en un point indique la direction de plus grande descente de la fonction et donc, on l'espère, la direction du minimum. Pour notre implantation, nous avons choisi d'utiliser l'algorithme de gradients conjugués. Cet algorithme fonctionne itérativement : il calcule d'abord le gradient au point courant, puis fait un pas dans une certaine direction et aboutit ainsi à un nouveau point qui devient le point courant. Il calcule alors le gradient à ce point et recommence le processus jusqu'à ce qu'il approche suffisamment un minimum, c'est-à-dire un point où le gradient est zéro. La direction du pas choisi par l'algorithme est calculée en combinant le gradient en ce point avec les gradients aux points précédents, d'où le nom de gradients conjugués.

Comme l'algorithme d'optimisation est itératif, il nous faut un point de départ pour démarrer le processus. Nous choisissons le vecteur $\boldsymbol{\lambda}$ initial en tirant de façon aléatoire pour chaque coefficient du vecteur un chiffre entre 0 et 1 selon une distribution uniforme. On calcule alors le vecteur de paramètres correspondant à cette valeur de $\boldsymbol{\lambda}$, puis on calcule le gradient pour ce $\boldsymbol{\lambda}$ et on lance l'optimisation.

Finalement, pendant le processus d'entraînement, après avoir déterminé le vecteur d'hyper-paramètres optimal, on recalcule le vecteur de paramètres en utilisant l'ensemble d'apprentissage au complet. Ceci n'est pas spécifié dans la description de l'algorithme, mais est utile en pratique. L'algorithme dit simplement de choisir les hyper-paramètres en optimisant le critère de sélection de modèle, et de choisir le vecteur de paramètres en minimisant le critère d'entraînement. Or, ceci demande de séparer l'ensemble d'apprentissage en deux parties, pour s'assurer que l'estimateur de l'erreur de généralisation est calculé en utilisant des exemples différents de ceux qui ont servis à calculer le vecteur de paramètres. Donc, une fois l'optimisation faite, le vecteur de paramètres dont nous disposons est celui qui correspond à une estimation des hyper-paramètres optimaux, mais il a été calculé en n'utilisant qu'une partie des exemples disponibles. Même avec la validation croisée, on n'utilise jamais tous les exemples d'apprentissage à la fois

pour calculer le vecteur de paramètres. On calcule un vecteur de paramètres pour chaque division de l'ensemble d'apprentissage, et donc tous les exemples servent à un moment ou un autre à calculer un vecteur de paramètres, mais on n'utilise jamais tous les exemples en même temps. Or, plus le nombre d'exemples d'apprentissage est grand, meilleure est la décision prise par un algorithme—du moins, en règle générale ; ceci est un fait bien connu de la théorie de l'apprentissage et est vrai non seulement pour notre algorithme mais aussi pour pratiquement tous les algorithmes adaptatifs. Donc, en recalculant le vecteur de paramètres à partir du vecteur d'hyper-paramètres optimal et de *tous* les exemples d'apprentissage, on obtient un vecteur qui devrait produire de meilleurs résultats.

6.6 Optimisation moyennée

L'algorithme présenté ci-haut pour optimiser les hyper-paramètres est très sensible aux variations dans les données. Deux ensembles d'apprentissage semblables produiront souvent deux vecteurs d'hyper-paramètres très différents. De plus, les hyper-paramètres produits par l'algorithme sont souvent plus petits que les véritables hyper-paramètres optimaux. Cette section présente une modification à l'algorithme dont le but est de résoudre ces deux problèmes.

La solution que nous proposons est de répéter plusieurs fois l'exécution de l'algorithme, en utilisant à chaque fois un ensemble d'apprentissage différent, obtenu en introduisant de petites variations dans l'ensemble initial. Chaque exécution produit un vecteur d'hyper-paramètres différent et on calcule la moyenne de ces vecteurs pour produire le vecteur final. L'idée est que chaque exécution peut produire de mauvaises valeurs pour certains des coefficients λ_m, mais pour un coefficient donné la plupart des exécutions produiront le bon résultat. Cette idée de générer plusieurs ensemble d'apprentissage différents à partir de l'ensemble initial est utilisée par plusieurs algorithmes d'apprentissage. On s'en sert entre autres pour diminuer la variance des résultats, comme dans notre cas, ou en-

core pour produire des meilleurs estimateurs de l'erreur de généralisation. Toutes ces variations sur un thème commun sont regroupées sous l'appellation générale **bootstrap** [8].

Le détail du nouvel algorithme est le suivant. D'abord, générer N ensembles d'apprentissage $D^1, \ldots, D^N$ en tirant aléatoirement avec remise $|D|$ exemples parmi D pour chacun. Appliquer ensuite l'algorithme d'optimisation simple des hyper-paramètres pour chacun des ensembles. Ceci produit N vecteurs d'hyper-paramètres $\boldsymbol{\lambda}^1, \ldots, \boldsymbol{\lambda}^N$. Ensuite, calculer la moyenne *géométrique* de ces hyper-paramètres selon la formule

$$\lambda_m^{\text{avg}} = \log(\frac{1}{N} \sum_n \exp \lambda_m^n) \tag{6.31}$$

Finalement, utiliser le critère de sélection de modèle C pour calculer les paramètres $\mathbf{b}^*(\boldsymbol{\lambda}^{\text{avg}}, D)$ correspondant aux hyper-paramètres moyennées. Ce vecteur est calculé en utilisant l'ensemble d'apprentissage initial ; on peut utiliser tous les exemples qui sont dans l'ensemble car on n'a pas à calculer l'estimateur de l'erreur de généralisation, puisque les hyper-paramètres sont déjà choisis.

Nous proposons d'utiliser la moyenne géométrique parce que celle-ci accorde plus d'importance aux coefficients élevés qu'aux petits coefficients. L'algorithme d'optimisation simple produit souvent des hyper-paramètres trop petits, et rarement des hyper-paramètres trop élevés. On veut donc compenser en privilégiant les valeurs élévées des coefficients quand on fait la moyenne. Ceci implique que parfois, une variable d'entrée sera pénalisée plus qu'elle ne devrait l'être, mais nous avons trouvé qu'en pratique la moyenne géométrique solutionne plus de problèmes qu'elle n'en crée.

Chapitre 7

Résultats expérimentaux

Ce chapitre décrit les expériences effectuées avec les algorithmes présentés dans les chapitres précédents. On peut diviser ces expériences en deux catégories :

- Les expériences de la première catégorie testent l'effet de chaque paramètre opérationnel de l'algorithme d'optimisation des hyper-paramètres. Nous avons ainsi expérimenté avec différentes tailles d'ensemble d'entraînement et différentes valeurs de K dans l'algorithme de validation croisée. Nous avons aussi testé des critères de sélection de modèle autres que la validation croisée afin de comparer leur performance.
- La deuxième catégorie regroupe les expériences qui comparent la performance des différents algorithmes. Cinq algorithmes ont été comparés : la régression linéaire classique, l'optimisation simple des hyper-paramètres, l'optimisation moyennée des hyper-paramètres, la régression pas à pas et la régression ridge adaptative.

Les expériences de la première catégorie sont présentés à la section 7.1, et les expériences de la deuxième, à la section 7.2.

7.1 Expériences préliminaires

7.1.1 Validation croisée *leave-one-out*

Cette première expérience compare deux critères de sélection de modèle. Le premier critère est l'estimateur de l'erreur de généralisation présenté à la section 2.3. Ce critère consiste à diviser l'ensemble d'apprentissage en deux sous-ensembles distincts : un ensemble d'entraînement et un ensemble de validation. Le vecteur de paramètres est sélectionné en minimisant le critère d'entraînement sur l'ensemble d'entraînement et le vecteur d'hyper-paramètres est sélectionné en minimisant le coût quadratique moyen sur l'ensemble de validation. Le deuxième critère est la validation croisée *leave-one-out*, c'est-à-dire la validation croisée avec $K = |D_{\text{ent}}|$.

Trois ensembles d'apprentissage ont été générés, de taille 20, 50 et 100 respectivement. Chaque exemple dans ces ensembles est constitué de 19 entrées et d'une sortie désirée. Les entrées ont été générées aléatoirement à partir de distributions normales. Deux groupes de distributions ont été utilisés pour générer les entrées. Le premier groupe P_1 était formé de cinq distributions normales $p_1^i(\mathbf{r})$, où $\mathbf{r}$ est un vecteur de taille 16 et $i = 1, \ldots, 5$. Les coefficients du vecteur de moyenne de chaque distribution p_1^i ont été fixés à zéro et chaque coefficient de la matrice de covariance de chacune des distributions a été généré aléatoirement à partir d'une normale $N(0, 1)$. Le deuxième groupe de distributions P_2 contenait trois distributions $p_2^j(\mathbf{s})$, où $\mathbf{s}$ est un vecteur de taille 3 et $j = 1, \ldots, 3$. La moyenne et la covariance de chacune de ces distributions ont été générées selon la même procédure que pour les distributions de P_1. Pour générer chaque exemple, on génère d'abord un premier groupe de seize entrées en choisissant une distribution p_1^i parmi P_1, et en générant un vecteur $\mathbf{r}$ à partir de cette distribution. On génère ensuite un vecteur $\mathbf{s}$ selon la même procédure avec P_2. Le vecteur d'entrées $\mathbf{x}$ est la concaténation de $\mathbf{r}$ et $\mathbf{s}$. La sortie désirée correspondante est calculée en additionant les trois coefficients du vecteurs $\mathbf{s}$ et en ajoutant un bruit Gaussien

	Taille de l'ensemble		
Critère	**20**	**50**	**100**
division simple	2.53	0.91	0.60
validation croisée	3.60	0.70	0.54

FIG. 7.1: Coût quadratique moyen obtenu en fonction de la taille de l'ensemble d'apprentissage pour deux critères de sélection de modèle : la division simple de l'ensemble d'apprentissage et la validation croisée *leave-one-out*.

$N(0, 1)$.

Pour chaque ensemble d'apprentissage, un ensemble de test de taille 10000 a été généré selon la même procédure. Pour mesurer la performance des algorithmes, nous avons calculé le coût quadratique moyen sur les exemples de l'ensemble de test. Chaque expérience a été répétée 100 fois et la moyenne des résultats a été calculée : le résultat est présenté à la figure 7.1 et est illustré à la figure 7.2. Remarquez ici qu'étant donné la grande taille de l'ensemble de test, la variance sur ces résultats est très petite. En fait, elle est pratiquement nulle.

Lorsque la taille de l'ensemble d'apprentissage est très petite, égale au nombre d'entrées de chaque exemple, les deux critères produisent de mauvais résultats. La division simple fait mieux que la validation croisée dans cette situation. Lorsque le nombre d'exemples d'apprentissage augmente, la performance des deux algorithmes s'améliore. La validation croisée produit de meilleurs résultats que la validation simple lorsque le nombre d'exemples est assez grand. Remarquez que comme il y a un bruit Gaussien de variance unitaire ajouté à la sortie désirée, le coût moyen minimal est de 0.5. Aucun algorithme ne pourrait faire mieux que ce minimum.

7.1.2 Influence de K sur la validation croisée

La deuxième expérience examine l'importance de K lorsque le critère de sélection de modèle est la validation croisée. Nous avons généré un ensemble

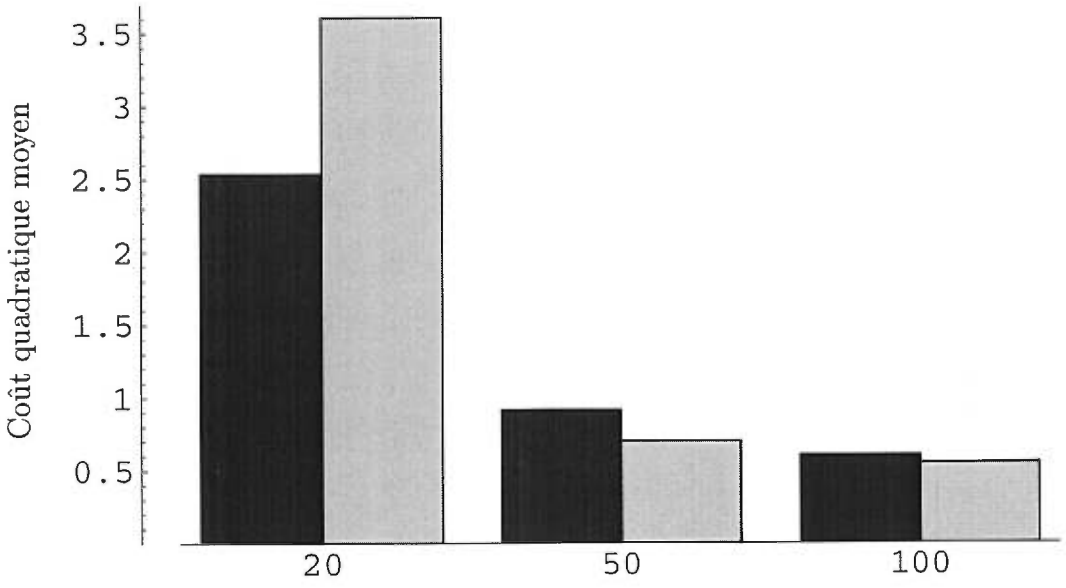


FIG. 7.2: Comparaison entre une division simple de l'ensemble d'apprentissage et la validation croisée, pour trois tailles différentes de l'ensemble d'apprentissage. Les colonnes noires indiquent le coût quadratique moyen pour la division simple de l'ensemble d'apprentissage. Les colonnes grises indiquent le coût moyen pour la validation croisée. Remarquez que pour le cas où l'ensemble d'apprentissage contient vingt exemples, le coût produit par les algorithmes est trop grand pour que le sommet des colonnes apparaissent sur le graphique.

d'apprentissage de taille 100 et un ensemble de test de taille 10000 selon la même procédure que pour l'expérience précedente. Nous avons comparé quatres critères de sélection de modèle sur ces données. Le premier critère était la division simple de l'ensemble d'apprentissage en deux sous-ensemble, comme précédemment. Les trois autres étaient la validation croisée avec $K = 10$, 20 et 100. Lorsque $K = 100$, on retrouve la validation croisée *leave-one-out* de la section précédente. L'expérience a été répétée cent fois et la moyenne a été calculée : le résultat est présenté à la figure 7.3

La validation croisée fait mieux que la division simple de l'ensemble d'ap-

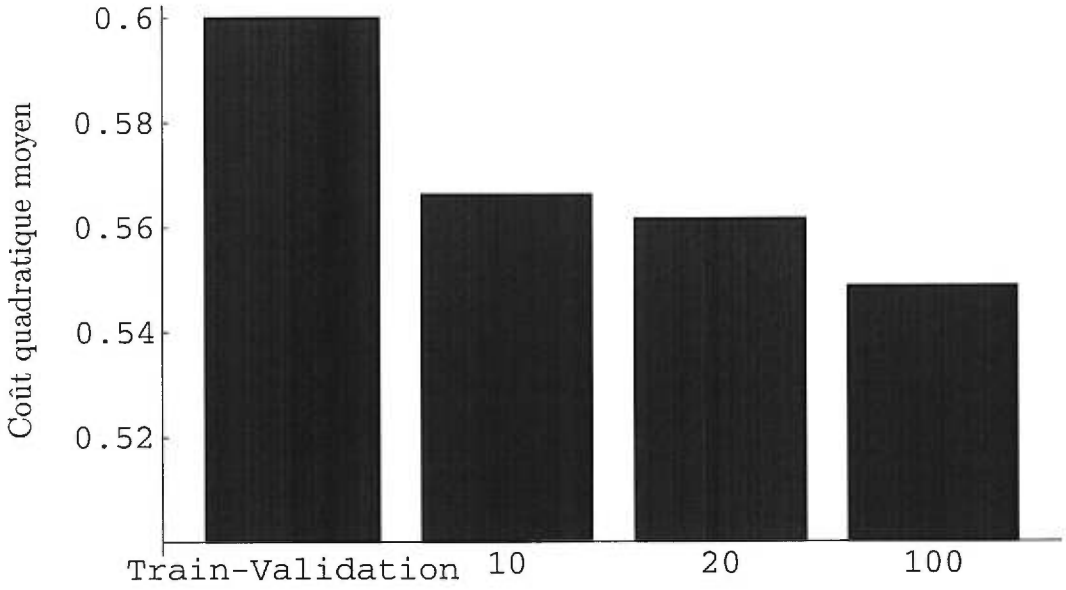


FIG. 7.3: Comparaison de quatre critères de sélection de modèle : la division simple de l'ensemble d'entraînement en deux sous-ensembles, et la validation croisée $K = 10$, 20 et 100.

prentissage pour toutes les valeurs de K. Plus K est grand, meilleure est la performance de l'algorithme d'optimisation. Remarquez qu'ici encore, la meilleure performance possible pour un algorithme est un coût quadratique moyen de 0.5, à cause du bruit sur la sortie désirée.

7.1.3 Estimateur 0.632 de l'erreur de généralisation

Cette expérience compare la validation croisée à un autre estimateur de l'erreur de généralisation : l'estimateur 0.632. Cet estimateur est en fait une somme pondérée de deux autres estimateurs : l'estimateur *bootstrap leave-one-out* et l'erreur empirique. L'estimateur 0.632 tire son nom de la pondération utilisée pour calculer la somme.

L'estimateur *bootstrap leave-one-out* est en quelque sorte une validation croisée *leave-one-out* moyennée. L'estimateur de l'erreur de généralisation produit par la

validation croisée *leave-one-out* est calculé en faisant la moyenne de T termes. Chacun de ces termes est obtenu en enlevant un des exemples $\mathbf{z}^t$ de l'ensemble d'apprentissage, en utilisant les $T-1$ exemples restants pour déterminer une droite de régression et en calculant l'erreur quadratique faite par cette droite sur l'exemple $\mathbf{z}^t$. L'estimateur *bootstrap leave-one-out* procède de façon similaire. Ici encore, l'estimateur est une moyenne de T termes. Seulement, pour chaque exemple $\mathbf{z}^t$, l'estimateur *bootstrap* produit R ensembles d'entraînements en tirant aléatoirement et avec remise $T-1$ exemples parmi les $T-1$ exemples restants dans l'ensemble d'apprentissage. Chacun de ces ensembles d'entraînement est utilisé pour calculer une droite de régression $\mathbf{b}^*$. L'estimateur de l'erreur de généralisation sur l'exemple $\mathbf{z}^t$ est la moyenne des erreurs quadratiques faites par les R droites de régression. Mathématiquement, l'estimateur *bootstrap leave-one-out* s'écrit

$$E(\boldsymbol{\lambda}, D) = \frac{1}{T}\sum_t \frac{1}{R}\sum_r Q(\mathbf{b}^*(\boldsymbol{\lambda}, D^r_{\text{ent}}), \mathbf{z}^t), \tag{7.1}$$

où D^r_{ent} est le r-ième ensemble d'entraînement obtenu à partir de $D\backslash\{\mathbf{z}^t\}$.

Comme les exemples sont tirés avec remise, pour chaque entraînement, le nombre d'exemples distincts dans l'ensemble d'entraînement sera plus petit que $T-1$. A cause de ceci, l'erreur quadratique faite sur chaque exemple $\mathbf{z}^t$ sera plus grande que l'erreur correspondante lorsque l'estimateur est la validation croisée. En effet, plus le nombre d'exemples d'entraînement est grand, meilleure est la droite de régression. La validation croisée *leave-one-out* utilise les $T-1$ exemples disponibles pour calculer la droite de régression, alors que l'estimateur *bootstrap leave-one-out* n'en utilise qu'une partie. L'estimateur de l'erreur de généralisation produit par le *bootstrap leave-one-out* sera donc plus grand que l'estimateur produit par la validation croisée *leave-one-out* pour un même ensemble d'apprentissage. On dira donc que l'estimateur *bootstrap* est un estimateur pessimiste de l'erreur de généralisation. Toutefois, en contrepartie, comme l'estimateur *bootstrap* est la moyenne de l'erreur faite par plusieurs droites de régression similaires, la variance de cet estimateur sera plus petite que celle de la validation croisée.

En d'autres termes, l'estimateur *bootstrap* est moins sensible aux variations dans l'ensemble d'apprentissage que ne l'est la validation croisée.

Pour tenter de profiter de l'avantage d'une petite variance tout en évitant un estimateur trop pessimiste, l'estimateur 0.632 combine l'estimateur *bootstrap leave-one-out* avec l'erreur empirique sur l'ensemble d'apprentissage. L'équation de l'estimateur 0.632 est

$$E_{0.632}(\boldsymbol{\lambda}, D) = wE_{\text{boot}} + (1 - w)E_{\text{re}} \tag{7.2}$$

On peut montrer (voir [8]) que lorsque $w = 2/3$, l'espérance de l'estimateur 0.632, lorsque calculée sur tous les ensembles d'apprentissage D possibles, est égale à l'espérance de l'erreur de généralisation, en supposant que les données sont effectivement linéaires avec un bruit normal de variance constante. Des calculs complexes suggèrent quant à eux que $w = 1 - e^{-1} = 0.632$ est le meilleur choix pour w. C'est cette dernière valeur qui est utilisée en pratique.

Pour cette expérience, nous avons généré des données selon le même modèle que pour les deux expériences précédentes. Nous avons généré un ensemble d'apprentissage de taille 100 et un ensemble de test de taille 10000. L'expérience a été répétée 100 fois. La moyenne des résultats est présentée à la figure 7.4. Les deux critères de sélection de modèle, validation croisée *leave-one-out* et l'estimateur 0.632, sont comparés aux résultats obtenus avec la régression linéaire classique. Comme on peut le voir, l'estimateur 0.632, lorsqu'utilisé comme critère de sélection de modèle dans notre algorithme d'optimisation des hyper-paramètres, ne permet pas de faire mieux que la régression linéaire classique. La validation croisée y est de beaucoup supérieure.

7.2 Comparaison des algorithmes

Cette section présente les résultats d'expériences visant à comparer les divers algorithmes décrit dans ce mémoire. La section 7.2.1 présente la structure de

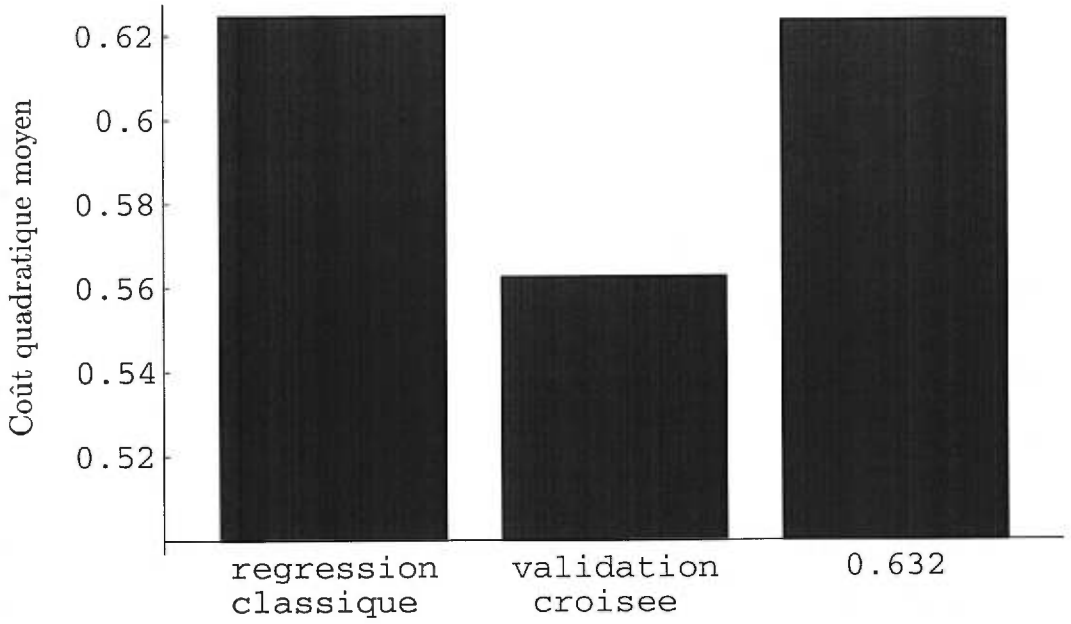


FIG. 7.4: Comparaison entre la régression linéaire classique, la validation croisée *leave-one-out* et l'estimateur 0.632.

simulation qui a été utilisée pour générer les données. Les deux sections suivantes, 7.2.2 et 7.2.3, présentent les résultats d'expériences sur des données en basse dimension et en haute dimension, respectivement. Finalement, la section 7.2.4 discute quelque peu informellement les résultats obtenus.

7.2.1 Structure de simulation

Pour comparer les performances des algorithmes, nous avons utilisé la structure de simulation proposée par Breiman [7]. Cette structure permet de tester des algorithmes de régression linéaire avec différentes distributions des données.

La structure est la suivante. Chaque exemple d'apprentissage est constitué de M entrées x_m et d'une sortie désirée y. Les M variables d'entrées sont générées selon une distribution normale de moyenne zéro et dont les coefficients de la matrice de covariance sont $\Sigma_{ij} = \rho^{|i-j|}$. Les sorties désirées sont calculées selon

l'équation

$$y = \sum_m \beta_m x_m + \epsilon \tag{7.3}$$

où les β_m sont les coefficients du vecteur de paramètre qui devra être estimé par la régression, et où ϵ est un bruit Gaussien distribué selon une normale $N(0,1)$. Les coefficients β_m sont calculés à l'aide de l'équation

$$\beta_m = \sum_i C[(h - |m - k_i|)^+]^2 \tag{7.4}$$

Le vecteur $\boldsymbol{\beta}$ est constitué d'un certain nombre de noyaux k_i. Plus un coefficient est "proche" d'un noyau, plus sa valeur sera élevée. La variable h détermine "l'écart" minimal entre un coefficient et un noyau pour que ce coefficient soit différent de zéro. Un exemple de vecteur $\boldsymbol{\beta}$ est donné à la section suivante. La constante C est choisie de telle sorte que le coefficient de corrélation R^2 soit approximativement 0.75.

Pour chaque expérience, on fixe les valeurs des noyaux k_i, ce qui laisse deux variables, ρ et h, que l'on peut faire varier pour générer des données issues de distributions différentes. La variable ρ contrôle le niveau de corrélation entre les entrées, tandis que la variable h détermine le nombre de coefficients β_m qui sont différents de zéro. Normalement, on s'attend à ce que certains algorithmes fonctionnent mieux lorsque la corrélation entre les entrées est faible, et d'autres lorsque la corrélation est élévée. De même pour le nombre de coefficients non nuls. On sait par exemple que les algorithmes qui font de la sélection de variables ont une meilleure performance lorsqu'il n'y a que quelques coefficients différents de zéro, et que ces coefficients sont très grands.

7.2.2 Expérience 1 : petit nombre d'entrées

Cette expérience utilise la structure décrite à la section précédente. Le nombre d'entrées M est fixé à 30 et les coefficients β_m sont calculés à partir de trois noyaux : $k_1 = 5$, $k_2 = 15$ et $k_3 = 25$. Trois valeurs différentes de ρ et trois valeurs différentes de h ont été testées, pour un total de neuf modèles différents. Les trois

valeurs de ρ sont 0.1, 0.5 et 0.9, et correspondent à une faible corrélation entre les données, une corrélation moyenne et une corrélation élevée. Les trois valeurs de h sont 1, 3 et 5, ce qui représente respectivement 3, 15 et 27 coefficients non nuls. Les vecteurs $\boldsymbol{\beta}$ pour chacune de ces trois valeurs sont illustrés à la figure 7.5.

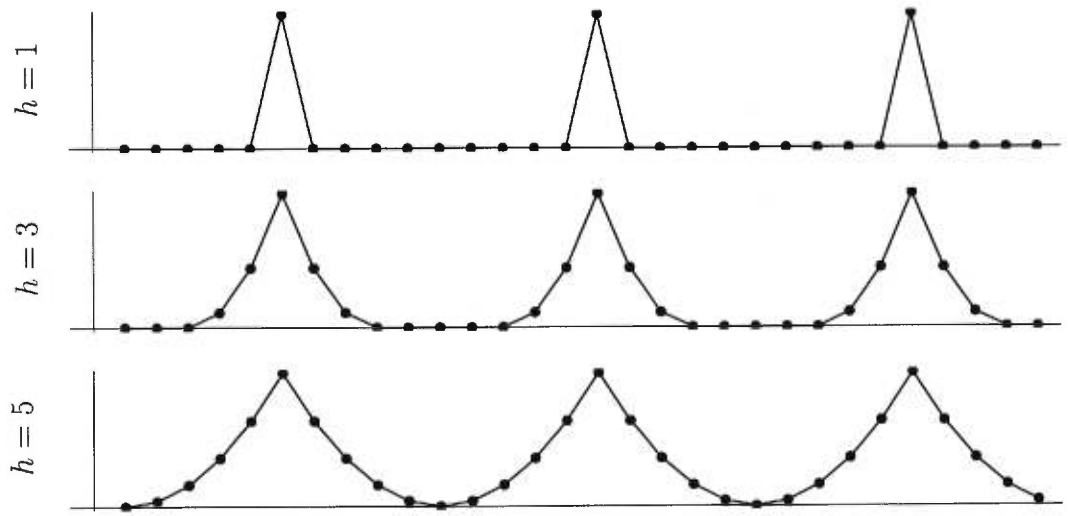


FIG. 7.5: Coefficients des trois vecteurs $\boldsymbol{\beta}$ utilisés pour générer les données de l'expérience 1. Les trois vecteurs sont générés à partir de trois valeurs différentes de la variable h.

Chacune des trois figures montre la taille relative de chacun des trente coefficients β_m par rapport aux autres. La valeur exacte de chacun des coefficients dépend de la constante C. Cette constante prend une valeur différente pour chaque modèle, tel qu'expliqué à la section 7.2.1.

Pour chacun des neuf modèles, un ensemble d'apprentissage de taille 60 et un ensemble de test de taille 10000 ont été générés. Chaque algorithme a été entraîné avec l'ensemble d'apprentissage et sa performance a été mesurée sur les exemples de l'ensemble de test. La mesure de performance utilisée est la moyenne des coûts quadratiques, donnée par l'équation 2.3. Remarquez que comme le bruit sur les exemples a une variance de un, le coût moyen optimal que l'on peut espérer obtenir est 0.5. La procédure d'entraînement et de test a été répétée cent fois, et la

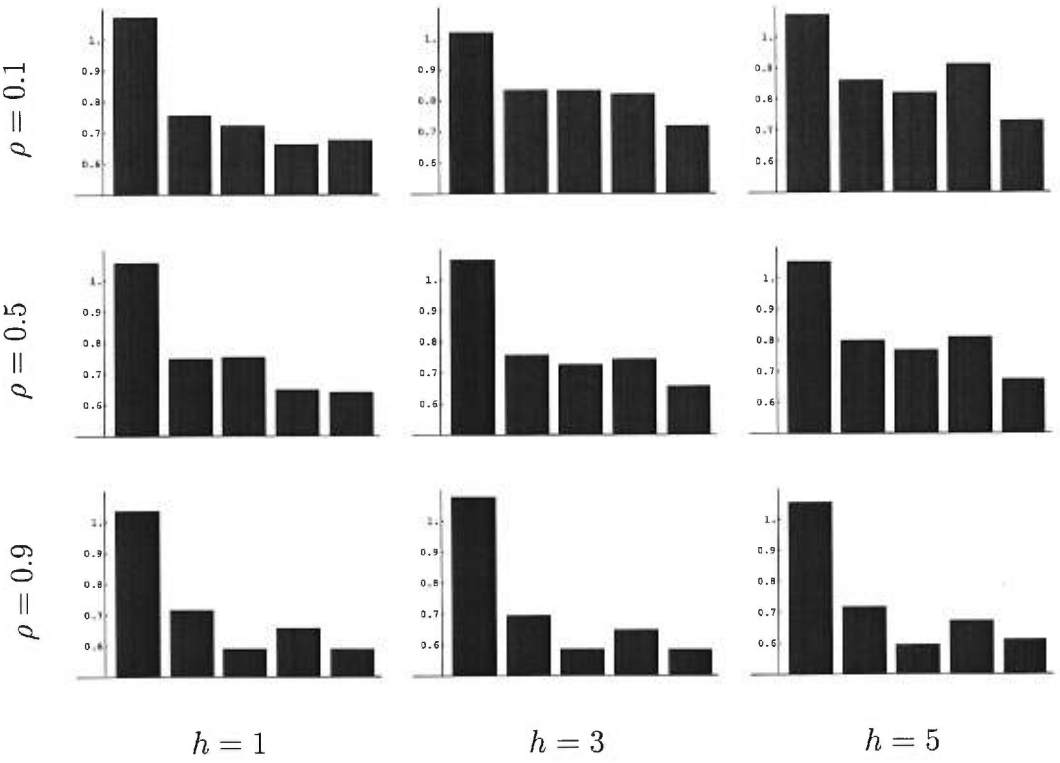


FIG. 7.6: Moyenne des coûts quadratiques des cinq algorithmes pour différentes valeurs de ρ et h. Pour chacun des neuf histogrammes, les cinq barres représentent, dans l'ordre de gauche à droite : la régression linéaire classique, l'optimisation simple des hyper-paramètres, l'optimisation moyennée des hyper-paramètres, la régression pas à pas et la régression ridge adaptative. L'erreur minimiale pour une régression linéaire pour chacun des modèles est de 0.5.

moyenne des cent exécutions a été calculée. Pour tous les algorithmes qui utilisent la validation croisée, nous avons utilisé $K = 10$. Pour l'algorithme d'optimisation moyennée des hyper-paramètres, la moyenne a été calculée sur dix échantillonages des données. Les résultats sont présentés à la figure 7.6.

Discussion des résultats

La première chose à remarquer est que tous les algorithmes font mieux que la régression linéaire classique. Aussi, aucun algorithme n'est supérieur aux autres

dans toutes les situations. En examinant la figure 7.6 du haut vers le bas, on remarque que tous les algorithmes performent mieux lorsque la valeur de ρ est grande. En l'examinant de la droite vers la gauche, on remarque également que tous performent mieux lorsque h est petit, c'est-à-dire lorsque le nombre de coefficients non nuls est petit. L'exception à cette règle est le cas où $\rho = 0.9$ et $h = 5$, haute corrélation et grand nombre de coefficients non nuls, où la plupart des algorithmes ont produit leurs meilleurs résultats.

Quand la corrélation entre les entrées est petite ou moyenne, la régression ridge adaptative est l'algorithme qui a produit les meilleurs résultats pour toutes les valeurs de h. La seule exception est le cas où $\rho = 0.1$ et $h = 1$: dans ce cas la régression pas à pas produite les meilleurs résultats. Ceci n'est pas très surpreneant puisque c'est un fait déjà connu que la sélection de variables fonctionne particulièrement bien dans ce cas.

Lorsque la corrélation est grande, l'algorithme d'optimisation moyennée des hyper-paramètres fait aussi bien ou mieux que la régression ridge adaptative. Notez aussi que l'optimisation moyennée fait aussi bien ou mieux que l'optimisation simple dans tous les cas, mais l'amélioration des performances est particulièrement importante quand la corrélation entre les entrées est élevée.

Niveau de signification des résultats

Considérons chacun des neuf modèles de données séparément. Pour chaque modèle, dénotons A_0 l'algorithme qui a produit les meilleurs résultats pour ce modèle, et dénotons A_i, $i = 1, \ldots, 4$, les quatres autres algorithmes. Pour chacune des cents répétitions de l'expérience, chaque algorithme A_i, $i = 0, \ldots, 4$, produit un coût Q_j^i pour ce modèle, où $j = 1, \ldots, 100$. Dénotons D_j^i la différence entre le coût de l'algorithme A_i et celui de l'algorithme A_0 pour la j-ième répétition de l'expérience.

$$D_j^i = Q_j^i - Q_j^0 \tag{7.5}$$

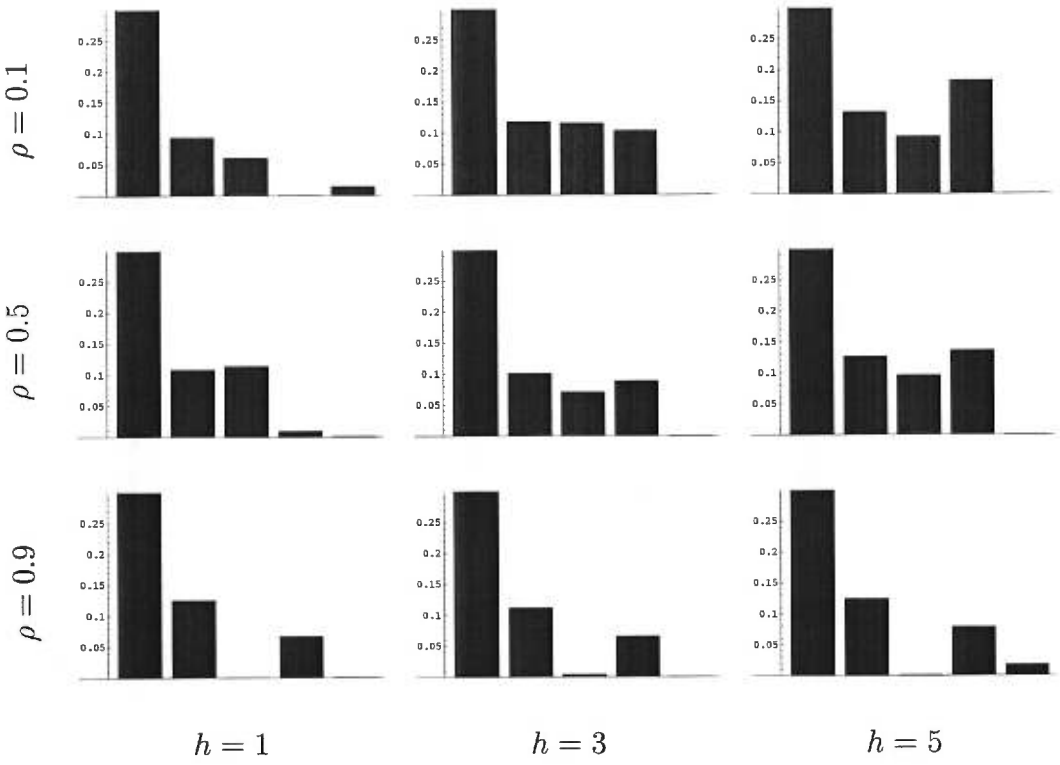


FIG. 7.7: Moyenne des différences entre le coût de chaque algorithme et le coût du meilleur algorithme pour ce modèle.

Définissons maintenant $\bar{D}^i$ comme la moyenne de ces différences pour les cents répétitions :

$$\bar{D}^i = \frac{1}{100} \sum_j D^i_j. \tag{7.6}$$

La moyenne $\bar{D}^i$ pour chaque algorithme et chaque modèle est présentée à la figure 7.7. Les algorithmes pour chaque modèle y sont présentés dans le même ordre que pour la figure 7.6.

Pour chaque moyenne $\bar{D}^i$, on peut calculer l'écart type qui lui est associé. Dénotons par S^i cet écart type.

$$S^i = \left(\frac{1}{99 \times 100} \sum_j (D^i_j - \bar{D}^i)^2 \right)^{\frac{1}{2}} \tag{7.7}$$

Si la différence moyenne $\bar{D}^i$ entre le coût l'algorithme A_i et le coût du meilleur algorithme pour ce modèle se situe à moins de deux écarts types de zéro, alors

on considère que la différence entre les performances des deux algorithmes n'est pas statistiquement significative. En d'autres termes, si l'axe $\bar{D}^i = 0$ se trouve à l'intérieur de deux écarts types de la valeur calculée de $\bar{D}^i$, la différence n'est pas significative.

Les différences moyennes et les écart types correspondants sont présentés à la figure 7.8. Cette figure est similaire à la figure 7.7, mais avec une échelle différente pour mettre en évidence les petites valeurs $\bar{D}^i$. Le sommet de chaque colonne indique la différence moyenne entre le coût d'un algorithme et le coût du meilleur algorithme pour ce modèle. La partie grise de chaque colonne couvre une région correspondant à deux écarts types à partir du sommet de la colonne. Cette région va donc du sommet D^i de la colonne jusqu'au point $D^i - 2 \times S^i$ qui se trouve à deux écarts types sous le sommet. Remarquez qu'à cause de l'échelle, le sommet la plupart des colonnes n'est pas visible sur les graphiques. De même, la plupart des régions marquant les écarts types sont trop loins de zéro pour apparaître sur la figure. Seules les colonnes qui représentent de petites différences sont totalement visibles.

On remarque que cinq différences apparaissent non-significatives. Pour le modèle $\rho = 0.1$ et $h = 1$, la régression pas à pas n'est pas significativement meilleure que la régression ridge adaptative. Dans le cas où $\rho = 0.5$ et $h = 1$, la régression ridge adaptative ne fait pas significativement mieux que la régression pas à pas. Enfin, pour les trois modèles avec $\rho = 0.9$, la différence entre la régression ridge adaptative et l'optimisation moyennée des hyper-paramètres n'est pas non plus significative.

Une deuxième façon de mesurer le niveau de signification des résultats est de calculer la p-valeur de la moyenne des différences. La p-valeur indique la probabilité d'observer le résultat obtenu, ou plus grand, si en fait les deux algorithmes comparés sont égaux. Pour calculer cette probabilité, on suppose que si les modèles sont égaux, alors la moyenne $\bar{D}^i$ des différences entre les deux modèles est une variable aléatoire distribuée selon une loi normale de moyenne

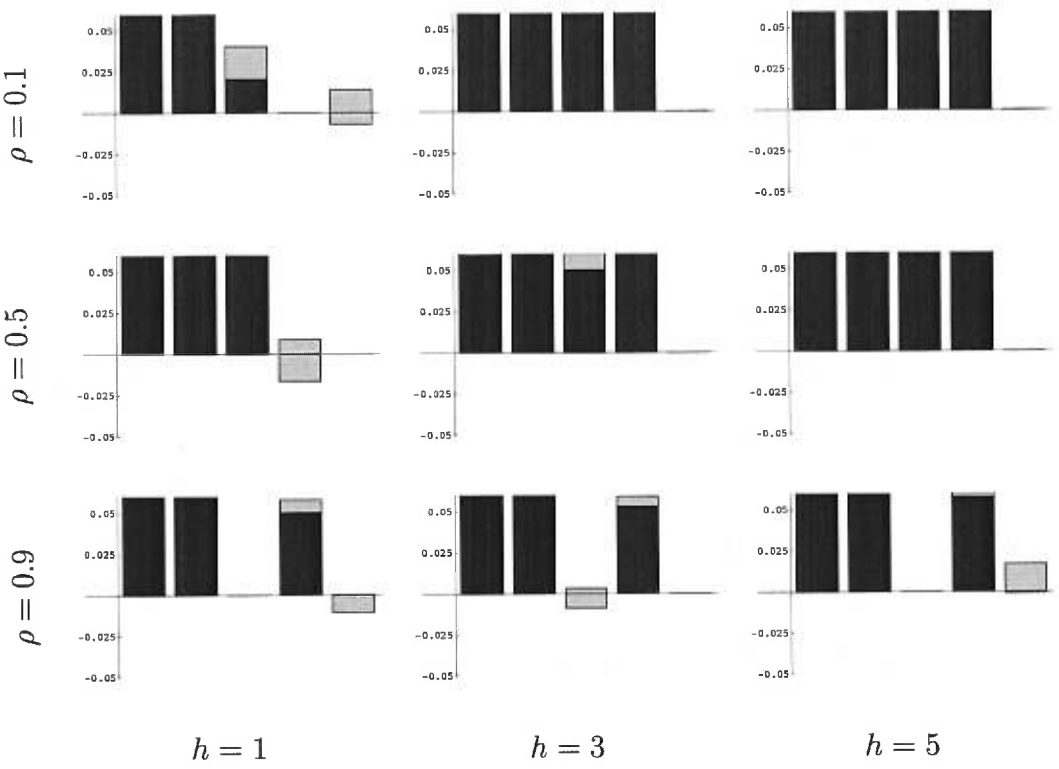


FIG. 7.8: Différence moyenne entre le coût d'un algorithme et le coût du meilleur algorithme pour ce modèle, et écart type des différences. Les colonnes grises démarquent la région qui se trouve à moins de deux écarts types de la différence moyenne.

zéro et de variance $V^i = S^i \times S^i$. La variable aléatoire

$$Z = \frac{\bar{D}^i}{S^i} \tag{7.8}$$

devrait donc être distribuée selon une normale $N(0,1)$. La p-valeur est la probabilité d'avoir observé la valeur de Z qui a été obtenue empiriquement ou une valeur plus grande.

Les p-valeurs obtenues pour les neuf modèles sont présentées à la figure 7.9. Dans trois cas, on observe une p-valeur de plus de dix pourcent. Ces trois cas font partie des cinq différences non-significatives que l'on obtenait lorsqu'on utilisait les écarts types pour estimer le niveau de signification des résultats.

Remarquez que l'analyse que nous venons de faire consiste en des comparai-

ϵ ϵ ϵ — 0.09	ϵ ϵ ϵ ϵ —	ϵ ϵ ϵ ϵ —
ϵ ϵ ϵ 0.24 —	ϵ ϵ ϵ ϵ —	ϵ ϵ ϵ ϵ —
ϵ ϵ — ϵ 0.48	ϵ ϵ 0.29 ϵ —	ϵ ϵ — ϵ 0.02

FIG. 7.9: Niveau de signification de la moyenne des différences entre chaque algorithme et le meilleur algorithme pour un modèle. Les expériences et les algorithmes sont présentés dans le même ordre que pour les figures 7.6, 7.7 et 7.8. Une p-valeur ϵ représente une p-valeur plus petite que 10^{-7}.

sons multiples. Les différences soulignées ne sont pas indépendantes les unes des autres, et les p-valeurs calculées non plus.

Conclusions

La régression ridge adaptative est l'algorithme qui produit le plus souvent les meilleurs résultats. Lorsque la corrélation entre les entrées est faible et que le nombre de coefficients non nuls est assez grand, il fait nettement mieux que les autres algorithmes. Dans le cas où la corrélation est faible et que le nombre de coefficients non nuls est petit, la régression pas à pas fait aussi bien que la régression ridge adaptative. Lorsque la corrélation entre les entrées est grande, alors l'algorithme d'optimisation moyennée des hyper-paramètres a une performance équivalente à celle de la régression ridge adaptative.

7.2.3 Expérience 2 : grand nombre d'entrées

Cette expérience reprend la structure des sections 7.2.1 et 7.2.2, mais cette fois le nombre d'entrée est fixé à 200. Ici encore, nous utilisons trois vecteurs de paramètres différents. Le premier vecteur contient seulement trois coefficients non nuls. Les premier, centième et deux centième coefficients sont fixés à un et les autres, à zéro. Les deux autres vecteurs sont générés selon la même méthode que

les vecteurs de l'expérience précédente. Les coefficients β_m de ces vecteurs sont calculés à partir de vingt noyaux. Le premier noyau est centré sur le cinquième coefficient et les noyaux subséquents sont répartis également à tous les dix coefficients jusqu'au cent quatre-vingt-quinzième. Nous utilisons deux valeurs de h, soit une pour chacun des deux vecteurs. Le premier vecteur est généré avec $h = 1$ et le deuxième avec $h = 3$. Ces deux valeurs correspondent respectivement à 20 et 100 coefficients non nuls. Comme à la section précédente, nous ferons référence aux vecteurs en utilisant la valeur de h qui a été utilisée pour les générer. Question de simplifier le texte, nous utiliserons $h = 0$ pour désigner le vecteur à trois coefficients non nuls, même si cette description n'est pas vraiment exacte. Les trois vecteurs de paramètres obtenus sont illustrés à la figure 7.10.

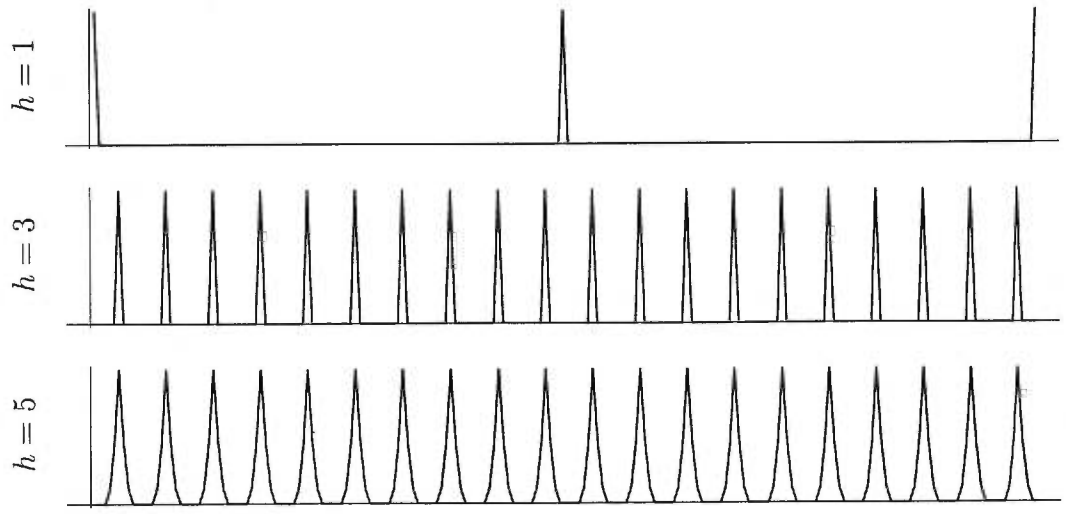


FIG. 7.10: Coefficients des trois vecteurs $\boldsymbol{\beta}$ utilisés pour générer les données de l'expérience 2. Les trois vecteurs sont générés à partir de trois valeurs différentes de la variable h.

Nous avons réutilisé les trois mêmes valeurs pour la variable ρ : 0.1, 0.5 et 0.9. Tout comme à la section précédente, nous avons donc neuf modèles de données différents. Pour chacun de ces neuf modèles, nous avons généré un ensemble d'apprentissage de taille 400 et un ensemble de test de taille 1000. Chaque expérience a été répétée cinq fois. Pour tous les algorithmes qui utilisent la validation croisée,

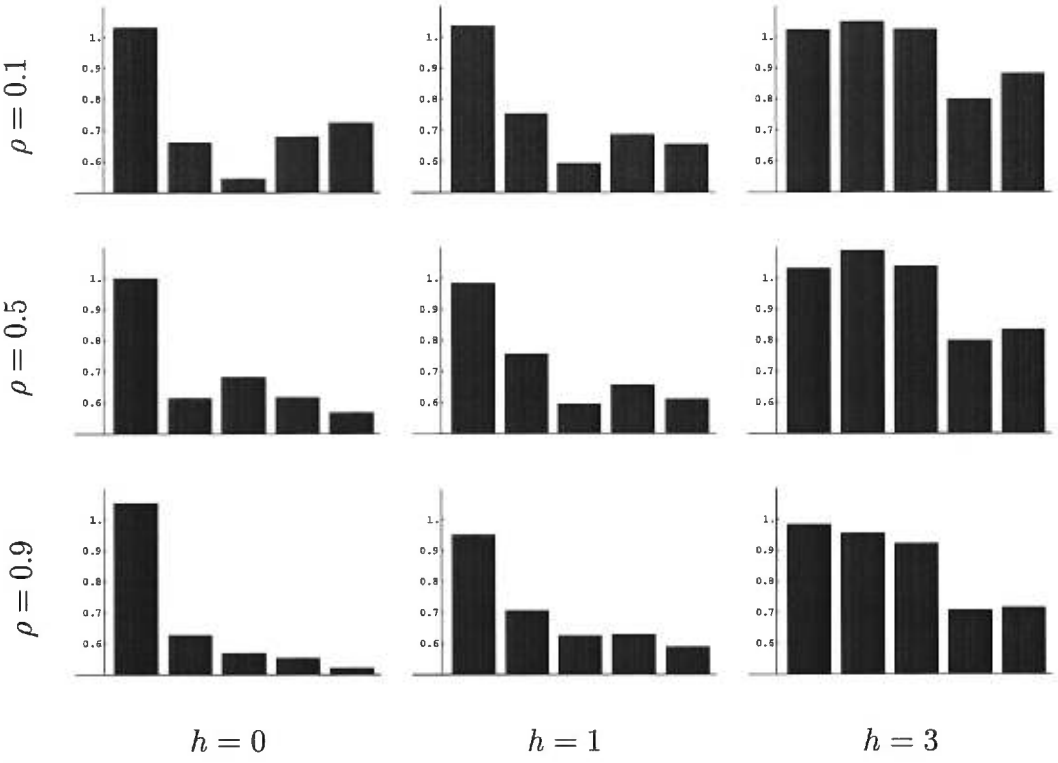


FIG. 7.11: Moyenne des coûts quadratiques des cinq algorithmes lorsque testés sur des données possédant un grand nombre d'entrées. Les cinq algorithmes sont, de gauche à droite : la régression linéaire classique, l'optimisation simple des hyper-paramètres, l'optimisation moyennée des hyper-paramètres, la régression pas à pas et la régression ridge adaptative.

nous avons utilisé $K = 10$. Pour l'algorithme d'optimisation moyennée des hyper-paramètres, la moyenne a été calculée sur dix échantillonages des données. Les résultats sont présentés à la figure 7.11.

Discussion des résultats

On remarque d'abord que tous les algorithmes font assez bien lorsque le nombre de coefficients non nuls est petit, mais que leur performance se détériore beaucoup lorsque le nombre de coefficients non nuls augmente. Les algorithmes font aussi légèrement mieux lorsque la corrélation entre les entrées est assez

grande.

Aucun algorithme ne fait mieux que les autres dans toutes les situations. Sur les neuf modèles, la régression pas à pas produit trois fois les meilleurs résultats, la régression ridge adaptative trois fois également, et l'optimisation moyennée trois fois aussi. La régression pas à pas est supérieure aux autres algorithmes lorsqu'il y a deux cents coefficients non nuls. Parmi les six modèles où le nombre de coefficients non nuls est de trois ou vingt, les deux autres algorithmes produisent la meilleure performance trois fois chacun.

Les résultats de la régression pas à pas sont un peu surprenants, puisque ce type d'algorithme est réputé pour offrir de bons résultats surtout lorsque le nombre de coefficients non nuls est petit et que la corrélation est faible. Ici, c'est le contraire qui se produit.

La performance de l'optimisation moyennée lorsque le nombre de coefficients non nuls est grand est décevante : dans deux cas, elle fait pire que la régression linéaire classique. L'optimisation simple quant à elle fait encore pire. L'optimisation moyennée fait tout de même très bien lorsque seulement quelques-unes des variables d'entrées sont corrélées avec la sortie. Elle apparaît donc particulièrement efficace pour éliminer de grandes quantités de bruit dans les variables d'entrées.

Niveau de signification des résultats

Nous avons encore une fois calculé l'écart type des moyennes des différences et les p-valeurs qui y sont associées. Nous présentons les écarts types dans le même format qu'à la section précédente. La figure 7.12 illustre les résultats.

Dans huit cas, la moyenne des différences se trouve à moins de deux écarts types de zéro. Lorsque $\rho = 0.5$ et $h = 0$, la régression ridge adaptative ne fait pas significativement mieux que la régression par étape, que l'optimisation simple des hyper-paramètres, ou de l'optimisation moyennée des hyper-paramètres. Lorsque $\rho = 0.5$ et $h = 1$ ou $h = 3$, l'optimisation moyennée n'est pas significativement meilleure que la régression ridge adaptative. Pour le modèle $\rho = 0.9$ et $h = 1$, la

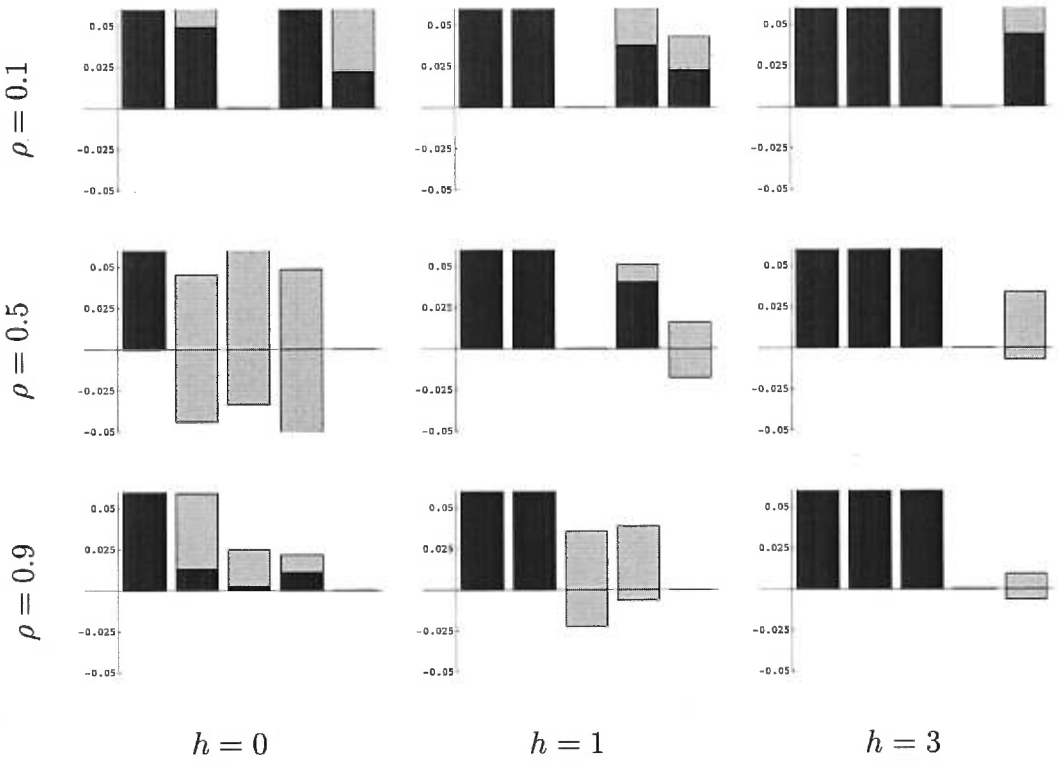


FIG. 7.12: Différence moyenne entre le coût d'un algorithme et le coût du meilleur algorithme pour ce modèle, et écart type des différences. Les colonnes grises démarquent la région qui se trouve à moins de deux écarts types de la différence moyenne.

régression ridge adaptative ne fait pas significativement mieux que la régression pas à pas ou que l'optimisation moyennée. Finalement, quand $\rho = 0.9$ et $h = 3$, les performances de la régression pas à pas et de la régression ridge adaptative ne sont pas significativement différentes. Le calcul des p-valeurs produit des résultats similaires, tel qu'indiqué à la figure 7.13.

En tenant compte du niveau de signification statistique suggéré par l'analyse des écarts types, on peut dire que la régression ridge adaptative produit six fois les meilleurs résultats, seul ou à égalité avec un autre algorithme, la régression pas à pas cinq fois, l'optimisation moyennée cinq fois, et l'optimisation simple une fois.

ϵ	ϵ	—	ϵ	0.01	ϵ	ϵ	—	ϵ	ϵ	ϵ	ϵ	ϵ	—	ϵ
ϵ	0.15	0.06	0.18	—	ϵ	ϵ	—	ϵ	0.17	ϵ	ϵ	ϵ	—	0.05
ϵ	0.01	0.02	ϵ	—	ϵ	ϵ	0.10	0.04	—	ϵ	ϵ	ϵ	—	0.12

FIG. 7.13: Niveau de signification de la moyenne des différences entre chaque algorithme et le meilleur algorithme pour un modèle. Les expériences et les algorithmes sont présentés dans le même ordre que pour les figures 7.6, 7.7 et 7.8. Ici encore, une p-valeur de ϵ représente une p-valeur plus petite que 10^{-7}.

Conclusions

Face à un problème de régression avec un grand nombre d'entrées, si aucune information n'est disponible sur les données, le choix du meilleur algorithme portera sur la régression ridge adaptative, puisqu'il offre les meilleures performances deux fois sur trois. Si toutefois on sait qu'un petit nombre d'entrées ne sont pas corrélées avec la sortie, mieux vaut utiliser la régression pas à pas pour les éliminer. L'optimisation moyennée des hyper-paramètres est quant à elle un choix valable lorsque le nombre le nombre d'entrées indépendantes de la sortie est assez grand.

Il faut toutefois remarquer que ces résultats ont été obtenus en ne répétant les expériences que cinq fois chacune. Le temps de calcul imposant que nécessite chaque expérience—plus de sept heures chacune—jumelé au nombre relativement élevé de modèles nous empêche de répéter les expériences assez souvent pour obtenir des résultats totalement concluants. Les tests statistiques que nous avons utilisés indiquent cependant que même avec seulement cinq répétitions, la plupart des résultats sont significatifs.

7.2.4 Discussion

L'algorithme d'optimisation des hyper-paramètres calcule le vecteur de paramètres **b** de la droite de régression en minimisant la même fonction de coût que la régression ridge adaptative. Toutefois, tandis que la régression ridge adaptative impose une contrainte sur les valeurs possibles des λ_m, l'optimisation des hyper-paramètres permet aux coefficients d'adopter n'importe quelle valeur. Peut-être serait-on alors portés à croire que l'optimisation des hyper-paramètres devrait faire mieux que la régression ridge adaptative. Après tout, s'il existe une valeur optimale des hyper-paramètres, cette valeur ne satisfait pas nécessairement la contrainte qu'impose la régression ridge adaptative. Théoriquement, l'algorithme d'optimisation des hyper-paramètres devrait être en mesure de trouver la valeur des λ_m qui minimise la fonction de coût, tandis que la régression ridge adaptative doit se contenter de choisir les hyper-paramètres les plus près des hyper-paramètres optimaux parmi ceux qui satisfont la contrainte. Mais tel n'est pas le cas. Pourquoi les résultats de la régression ridge adaptative sont-ils supérieurs ?

Le premier point à souligner est que l'algorithme d'optimisation des hyper-paramètres ne parvient jamais à trouver la valeur optimale des hyper-paramètres. La valeur du vecteur d'hyper-paramètres $\boldsymbol{\lambda}^*(D)$ qui minimise l'erreur sur l'ensemble d'apprentissage D n'est pas exactement la même que celle du vecteur $\boldsymbol{\lambda}^*$ qui produit la plus petite erreur de généralisation. Tel qu'expliqué au chapitre 2, $\boldsymbol{\lambda}^*(D)$ est un estimateur bruité de $\boldsymbol{\lambda}^*$. De plus, plus la taille de l'ensemble d'apprentissage est petite, plus le niveau de bruit sera élevé. Or, le problème qui nous intéresse est justement le cas où le nombre d'exemples est petit. Le bruit sur la valeur calculée des hyper-paramètres sera donc important et cette valeur sera différente du véritable optimum.

Un deuxième problème provient du fait que l'algorithme d'optimisation des hyper-paramètres peut résulter en un minimum *local* du critère de sélection de modèle, et pas nécessairement un minimum *global*. Le vecteur d'hyper-paramètres

produit n'est donc pas nécessairement le vecteur optimal.

Il est donc naïf de croire que l'algorithme d'optimisation des hyper-paramètres parviendra à trouver la meilleure valeur possible pour le vecteur $\boldsymbol{\lambda}$. Ceci n'explique toutefois pas pourquoi la valeur que cet algorithme produit est moins bonne que celle produite par la régression ridge adaptative. Pourquoi la régression ridge adaptative fait-il mieux ?

La régression ridge adaptative parvient à obtenir de bonnes performances en imposant une contrainte intelligente sur les hyper-paramètres. La contrainte n'est pas choisie uniquement pour faciliter la calcul des λ_m, mais aussi pour guider l'algorithme vers de meilleurs valeurs pour ces variables. Réalisant que le niveau de bruit est trop important pour que la minimisation de la fonction de coût puisse à elle seule produire les hyper-paramètres optimaux, la régression ridge adaptative restreint le domaine de la minimisation à une région de l'espace des λ_m où une bonne performance peut être atteinte. Le véritable point optimal n'appartient pas nécessairement à cette région, mais on peut néanmoins y trouver un grand nombre de vecteur $\boldsymbol{\lambda}$ qui permettront d'obtenir de bons résultats. On peut trouver une interprétation simple de la contrainte qu'utilise la régression ridge adaptative. L'équation 5.41 force la taille du vecteur de paramètres $\boldsymbol{\lambda}$ à être une fonction de l'hyper-paramètre μ. Ainsi, pour un μ donné, la taille du vecteur de paramètre est fixe. La régression ridge adaptative utilise alors l'ensemble d'apprentissage, via le critère d'entraînement, pour comparer les vecteurs $\boldsymbol{\lambda}$ ayant la même taille, et utilise le critère de sélection de modèle pour sélectionner la meilleure taille pour le vecteur de paramètres.

Trouver une telle contrainte n'est pas nécessairement évident pour tous les problèmes d'apprentissage supervisé. La régression linéaire cependant s'y prête particulièrement bien. L'algorithme classique a été étudié en détail depuis des dizaines d'années et les chercheurs ont ainsi acquis au fil des ans une excellente compréhension de son comportement sous diverses conditions expérimentales. De plus, la régression linéaire permet une interprétation géométrique simple, tel

qu'illustré au chapitre 5. Pouvoir ainsi visualiser l'espace des paramètres permet d'approfondir notre compréhension du problème et de trouver ainsi une contrainte utile pour guider l'algorithme.

L'algorithme d'optimisation des hyper-paramètres quant à lui ne fait usage d'aucune de ces deux ressources. Il s'agit plutôt d'une approche de type "force brute" qui ne tire aucunement partie de notre compréhension du problème. L'algorithme a toutefois un avantage important. Bien que nous n'en ayons que très peu fait mention jusqu'ici, l'optimisation des hyper-paramètres peut être appliqué non seulement à la régression linéaire, mais aussi à un grand nombre de problèmes d'apprentissage supervisé. La seule condition est que le problème puisse être exprimé en termes de critère d'entraînement et de critère de sélection de modèle. Cette condition est somme toute assez générale et permet que l'approche soit appliquée à un grand nombre de problèmes. Ainsi, elle a entre autres été developpée pour des réseaux de neurones, où elle permet de choisir la pénalité associée à chacun des poids dans le réseau [2].

En résumé, la régression ridge adaptative produit de meilleurs résultats que l'algorithme d'optimisation des hyper-paramètres parce qu'il utilise une contrainte sur les hyper-paramètres qui guide l'algorithme dans sa recherche des hyper-paramètres optimaux. Un algorithme comme la régression ridge adaptative qui sait tirer partie de notre compréhension d'un problème fera habituellement mieux qu'une approche de type force brute qui recherche aveuglément une solution. Toutefois, l'approche de type force brute, si elle est assez générale, pourra être appliquée à d'autres problèmes similaires, tandis que la compréhension du problème est quant à elle plus difficilement transférable. Il reste toutefois à prouver que l'algorithme d'optimisation des hyper-paramètres produit de bons résultats sur d'autres problèmes que la régression linéaire, notre recherche s'étant pour l'instant restreinte à ce problème particulier.

Chapitre 8

Conclusion

La régression linéaire classique produit de mauvais résultats lorsque la taille de l'ensemble d'entraînement est du même ordre que la taille du vecteur de paramètres de la droite de régression. Plusieurs algorithmes ont été proposés pour améliorer la régression classique dans cette situation. Ces algorithmes sont divisés en deux grandes catégories selon l'approche qu'ils utilisent.

La première catégorie regroupe les algorithmes qui font de la sélection de variables : sélection avant, élimination arrière, régression pas à pas et autres algorithmes du même genre. Ces algorithmes utilisent des tests statistiques pour mesurer l'utilité de chaque variable d'entrée dans la prédiction de la sortie. Les variables les moins utiles sont éliminées et la régression classique est appliquée avec les variables d'entrées restantes.

La deuxième catégorie d'algorithmes regroupe les algorithmes basés sur le principe de pénalisation. Deux de ces algorithmes présentés dans ce mémoire sont la régression ridge et la régression ridge adaptative. Ces algorithmes utilisent toutes les variables d'entrées disponibles, mais pénalisent les coefficients du vecteur de paramètres en fonction de leur taille. Un petit coefficient b_m signifie que la variable d'entrée x_m correspondante aura peu d'influence dans le calcul de la sortie. On peut ainsi utiliser un seul terme de pénalité regroupant toutes les variables, comme le fait la régression ridge, ou bien utiliser M termes de pénalité, un

pour chaque variable, ce que fait la régression ridge adaptative. Dans le deuxième cas, ceci revient à faire une sélection de variables "molle" : plutôt que d'inclure ou exclure totalement une variable d'entrée dans l'ensemble des variables utilisées par la régression, on lui assigne une pénalité qui est inversement proportionnelle à son "niveau d'inclusion".

Lorsque la pénalisation est utilisée pour améliorer la régression, chaque terme de pénalité est pondéré par une variable que l'on nomme hyper-paramètre. Pour une valeur donnée des hyper-paramètres, un algorithme produira toujours le même vecteur de paramètres $\mathbf{b}$: le vecteur de paramètres est ainsi une fonction des hyper-paramètres. Il existe une valeur optimale des hyper-paramètres qui produira le vecteur $\mathbf{b}$ avec la meilleure erreur de généralisation. Cette valeur optimale doit habituellement être déterminée par essai et erreur étant donné qu'il n'existe pas d'algorithme permettant de la calculer.

L'algorithme d'optimisation des hyper-paramètres que nous avons étudié utilise plusieurs termes de pénalité dans le calcul du vecteur de paramètres. Il détermine le vecteur $\mathbf{b}$ en minimisant par rapport aux coefficients b_m une fonction de coût qui est une fonction à la fois des paramètres et des hyper-paramètres. La valeur optimale des hyper-paramètres λ_m est calculée en minimisant un estimateur de l'erreur de généralisation, dérivable par rapport aux λ_m.

Cet algorithme diffère des algorithmes déjà proposés sur deux points importants. Les autres algorithmes qui utilisent des hyper-paramètres en utilisent habituellement un seul. De plus, la valeur de cet hyper-paramètre doit être déterminée par essai et erreur : il n'existe aucune technique générale pour calculer sa valeur optimale. Ceci est vrai non seulement en régression linéaire, mais pour tous les problèmes d'apprentissage supervisés, comme par exemple les réseaux de neurones. L'algorithme d'optimisation présenté dans ce mémoire utilise quant à lui plusieurs hyper-paramètres—un pour chaque variable d'entrée—et leur valeur peut être calculée en utilisant un algorithme d'optimisant numérique utilisant la dérivée d'un estimateur de l'erreur de généralisation.

Les résultats expérimentaux indiquent que parmi les algorithmes présentés, la régression ridge adaptative est celui qui produit les meilleurs résultats. L'optimisation des hyper-paramètres a la meilleure performance pour certaines distributions des données, mais ne parvient pas à faire mieux que la régression ridge adaptative en général. Sa performance est sensiblement équivalente à celle de la régression pas à pas, bien que les deux algorithmes obtiennent leurs meilleurs résultats dans des situations différentes.

Une hypothèse pour expliquer pourquoi la régression ridge adaptative produit de meilleurs résultats que l'algorithme d'optimisation des hyper-paramètres est que la régression ridge adaptative force le vecteur de paramètres à obéir à certaines contraintes, choisies spécifiquement pour guider l'algorithme dans la recherche du vecteur de paramètres optimal. L'algorithme d'optimisation des hyper-paramètres n'utilise pas de telles contraintes et recherche plutôt le vecteur optimal parmi tout l'ensemble des solutions possibles. Cette approche de type force brute est plus générale, mais moins efficace qu'une approche qui utilise notre compréhension d'un problème et qui est conçue spécifiquement pour résoudre ce problème particulier. Car, contrairement à la régression ridge adaptative qui sert uniquement à la régression linéaire, l'algorithme d'optimisation des hyper-paramètres peut être utilisé pour résoudre un grand nombre de problèmes, pourvu que ces problèmes puissent être exprimés en termes d'hyper-paramètres, de critère d'entraînement et de critère de sélection de modèle.

La suite logique de ce mémoire serait donc d'étudier plus en détails certains de ces problèmes et de comparer l'algorithme d'optimisation des hyper-paramètres aux méthodes qui sont actuellement utilisées pour les résoudre. Un bon point de départ serait d'utiliser l'algorithme pour déterminer la force de la pénalisation dans le "weight decay" des réseaux de neurones, ceci étant certainement une des applications les plus connues et les plus utilisées des hyper-paramètres. Les bases mathématiques pour cette application de l'algorithme d'optimisation ont déjà été développées dans [2], il reste maintenant à faire des expériences pour voir la

qualité des résultats produits.

Bibliographie

[1] E.M.L Beale. *Introduction to Optimization.* John Wiley and Sons, 1988.

[2] Y. Bengio. Continuous optimization of hyper-parameters. Technical Report 1144, Departement d'informatique et de recherche operationnnelle, Université de Montreal, 1999.

[3] C. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

[4] L. Bottou. Online algorithms and stochastic approximations. In D. Saad, editor, *Online Learning in Neural Networks.* Cambridge University Press, 1998.

[5] H. Boukari and Y. Grandvalet. Penalisation multiple adaptative. *13emes Journées Francophones sur l'Apprentissage*, 1998.

[6] L. Breiman. Better subset selection using the non-negative garotte. Technical report, University of California, Berkeley, 1993.

[7] L. Breiman. Bagging predictors. Technical report, Statistics Department, University of California at Berkeley, 1994.

[8] A.C. Davison and D.V. Hinkley. *Bootstrap methods and their application.* Cambridge University Press, 1997.

[9] N.R. Draper and H. Smith. *Applied Regression Analysis.* John Wiley and Sons, 1981.

[10] Y. Grandvalet. Least absolute shrinkage is equivalent to quadratic penalization. In M. Boden L. Niklasson and T. Ziemske, editors, *ICANN'98*, volume 1 of *Perspectives in Neural Computing*, pages 201–206. Springer, 1998.

[11] Y. Grandvalet and S. Canu. Outcomes of the equivalence of adaptive ridge with least absolute shrinkage. In S.A. Solla M.S. Kearns and S.A. Cohn, editors, *Advances in Neural Information Processing Systems 11*. MIT Press, 1999.

[12] G.E. Hinton. Learning translation invariant recognitionin massively parallel networks. In A.J. Nijman J.W. de Barker and P.C. Treleaven, editors, *Proceedings PARLE Conference on Parallel Architectures and Languages Europe*, pages 1–13. Springer-Verlag, Berlin, 1987.

[13] A.E. Hoerl. Application of ridge analysis to regression problems. *Chem. Eng. Prog.*, 58 :54–59, 1962.

[14] A.E. Hoerl and R.W. Kennard. Ridge regression : applications to nonorthogonal data. *Technometrics*, 12 :69–82, 1970.

[15] A.E. Hoerl and R.W. Kennard. Ridge regression : biased estimation for nonorthogonal problems. *Technometrics*, 12 :55–67, 1970.

[16] A.E. Hoerl and R.W. Kennard. A note on a power generalization of ridge regression. *Technometrics*, 17 :269, 1975.

[17] A.E. Hoerl and R.W. Kennard. Ridge regression : iterative estimation of the biasing parameter. *Communications in Statistics*, A5 :77–88, 1976.

[18] R.W. Kennard Hoerl, A.E. and K.F. Baldwin. Ridge regression : some simulations. *Communications in Statistics*, 4 :105–123, 1975.

[19] D.J.C. Mackay. Bayesian interpolation. *Neural Computation*, 4(3) :415–447, 1992.

[20] D.J.C. Mackay. The evidence framework applied to classification networks. *Neural Computation*, 4(5) :720–736, 1992.

[21] D.J.C. Mackay. Information-based objective functions for active data selection. *Neural Computation*, 4(4) :590–604, 1992.

[22] D.J.C. Mackay. A pratical bayesian framework for backpropagation networks. *Neural Computation*, 4(3) :448–472, 1992.

[23] D.J.C. Mackay. Bayesian methods for backpropagation networks. In J.L van Hemmen E. Domany and K. Schulten, editors, *Models of Neural Networks III*, chapter 6. Springer-Verlag, New York, 1994.

[24] D.J.C. Mackay. Hyperparameters : optimise or integrate out ? In G. Heidbreder, editor, *Maximum Entropy and Bayesian Methods, Santa Clara 1993.* Kluwer, 1994.

[25] D.J.C. Mackay. Bayesian neural networks and density networks. *Nuclear Instruments and Methods in Physics Research, A*, 354(1) :73–80, 1995.

[26] D.J.C. Mackay. Bayesian non-linear modelling for the 1993 energy prediction competition. In G. Heidbreder, editor, *Maximum Entropy and Bayesian Methods, Santa Clara 1993*, 1995.

[27] D.C. Montgomery and E.A. Peck. *Introduction to Linear Regression Analysis.* John Wiley and Sons, 1982.

[28] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in Fortran 77, Second Edition.* University of Cambridge, 1996.

[29] R.J. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society B*, 58 :267–288, 1995.

[30] V. Vapnik. *Estimation of Dependencies Based on Empirical Data.* Springer-Verlag, New-York, NY, USA, 1982.

[31] V. Vapnik. *The Nature of Statistical Learning Theory.* Springer, 1995.

[32] V. Vapnik. *Statistical Learning Theory.* Wiley, 1998.

[33] S. Weisberg. *Applied Linear Regression.* John Wiley and Sons, 1980.