Notes de cours de réseaux de neurones

Olivier François
LMC - IMAG, BP 53, 38041 Grenoble cedex 9
Olivier.Francois@imag.fr

29 mars 2001

Table des matières

Chapitre 1

Introduction

1.1 Problèmatiques abordées

Les méthodes neuronales se sont développées depuis ces trente dernières années simultanément au paradigme de l'apprentissage (*machine learning*). Selon ce paradigme, les machines ne sont pas programmées à l'avance pour une tâche donnée (par exemple, la reconnaissance d'une forme), mais "apprennent" à effectuer cette tâche à partir d'exemples.

Un problème typique d'apprentissage consiste à apprendre à prédire une variable Y, appelée réponse ou variable de sortie, à partir d'une variable X (appelée variable d'entrée ou vecteur de caractéristiques). En général, la relation entre Y et X est stochastique. Le couple (X, Y) obéit à une certaine loi de probabilité, notée p, et on dispose d'un *ensemble d'apprentissage* $(x_1, y_1), \ldots, (x_n, y_n)$ constitué de données indépendantes observées à partir de la loi p. Dans cet échantillon, y_i est la réponse correspondant à l'entrée x_i.

Dans un problème de classification entre deux classes, $y \in \{0, 1\}$ peut par exemple représenter la parité du vecteur d'entrée $x \in \{0, 1\}^\ell$. Dans ce cas, le problème est dégénéré au sens où y est uniquement déterminé par x. Mais la variable Y peut aussi coder l'identification de la voix humaine, lorsque x représente de manière appropriée un phonème précis. Dans ce cas, l'identification n'est pas certaine, et le problème est non-dégénéré.

Le problème d'apprentissage consiste à construire une fonction f (ou machine) à partir des données $(x_1, y_1), \ldots, (x_n, y_n)$ de sorte que $f(X)$ soit une bonne approximation de la réponse souhaitée Y. Souvent, on choisit f de manière à minimiser un critère fonctionnel. Par exemple, on peut former la somme des erreurs quadratiques

$$E = \sum_{i=1}^{n} (y_i - f(x_i))^2$$

et chercher à rendre cette somme la plus petite possible[1].

1. Attention à ne pas faire de confusion entre erreur E et espérance mathématique.

De tels problèmes de minimisation sont bien étudiés en statistique. Comme nous le verrons plus tard, il s'agit d'un moyen d'estimer la fonction de régression

$$g(x) = E[Y|X = x] .$$

Dans le cas d'une classification binaire, la fonction de régression se réduit tout simplement à

$$g(x) = P[Y = 1|X = x] .$$

Une telle grandeur n'est rien d'autre que la probabilité (conditionnelle) d'affecter x dans la classe à laquelle il appartient.

En fait, un problème d'apprentissage est un problème d'estimation fonctionnelle. Ce cours présente l'estimation et la modélisation (le choix d'un bon modèle) à l'aide de réseaux de neurones. Nous aborderons les questions pratiques à l'aide du logiciel S-Plus. L'utilisation pratique des réseaux de neurones demande une bonne expérience, car la plupart des techniques utilisées relèvent d'une combinaison entre des idées théoriques et d'autres plus empiriques qu'il est parfois difficile d'évaluer sans expérimentation. Il ne s'agit pas d'un cours de statistique proprement dite, car la méthodologie usuelle des tests ne sera pas utilisée (il serait possible de le faire). Il s'agit en fait d'un cours centré sur un objet difficile à maîtriser théoriquement, pour lequel des idées venant de la statistique (et des probabilités) sont très utiles. Ces idées apportent un éclairage sur les méthodes que l'on peut utiliser pour appréhender des problèmes concrets à l'aide de modèles neuronaux. Toutefois, il peut être utile de mentionner que la démarche adoptée est courante pour la plupart des modèles statistiques "non-paramétriques" et ne se limite pas à la classe de modèles étudiés ici (modèles additifs généralisés, projection pursuit, fenêtres de Parzen, méthodes de plus proche voisin, etc).

Ce cours est organisé de la manière suivante. Nous présentons tout d'abord les modèles que nous allons étudier dans la section 1.2. Il s'agit d'une classe restreinte de modèles neuronaux : les réseaux "feed-forward". Il existe une multitude d'autres modèles, dont la discussion mériterait beaucoup plus de temps que celui qui est imparti à ce cours. Puis nous présenterons la bibliothèque `nnet`, disponible sous S-Plus, pour ajuster des modèles neuronaux (section 1.2.1). Le chapitre 2 est consacré à la régression non-linéaire et le chapitre 3 aux problèmes de classification. Les deux approches sont très similaires, mais il nous a paru bon de les séparer. Les principaux concepts théoriques sont présentés pour la régression non-linéaire (plus simple), mais se généralisent aux problèmes de classification. Les applications réalistes des réseaux de neurones concernent toutefois le plus fréquemment des problèmes de classification (ou de reconnaissance des formes).

Ce cours s'inspire principalement de l'ouvrage de Ripley (1996) et des articles de Geman et al. (1992), et White (1989). La bibliothèque `nnet` est une bibliothèque libre d'accès programmée par Ripley.

1.2 Présentation des réseaux de neurones

Les modèles "feed-forward" de réseaux neuronaux que l'on appelle parfois *perceptrons multi-couches* ont été popularisés par Rumelhart, Hinton et Williams (1986). Ces modèles, dits flexibles, proposent une généralisation (non-linéaire) des modèles linéaires multidimensionnels connus.

FIG. 1.1 – *Liens entrées/sorties pour un réseau de neurones à une couche cachée*

Nous nous limiterons à la forme la plus simple (et la plus commune) de réseau de neurones. Le réseau se structure en trois couches successives. La couche d'entrée permet de saisir le vecteur de caractéristiques x, et la couche de sortie fournit la prédiction. Les entrées sont redistribuées sur des unités *cachées* intermédiaires à la sortie. Ces unités cachées forment la seconde couche. Un réseau de neurones est paramétré par un ensemble de *poids* w_i^j et de *biais* α_i. Nous noterons θ le paramètre global correspondant

$$\theta = (\alpha_h, w_h^j, \alpha^k, w_h^k).$$

Chaque unité effectue une somme pondérée de ses entrées, ajoute une constante et évalue le résultat selon une fonction ϕ_h. Les unités de sortie ont la même forme, mais l'évaluation se fait selon une fonction ϕ_o qui peut être différente de ϕ_h. Ainsi, le modèle calcule la fonction

$$g^k(x,\theta) = \phi_o \left(\alpha^k + \sum_h w_h^k \, \phi_h\Big(\alpha_h + \sum_j w_h^j \, x^j\Big) \right) \;, \quad 1 \leq k \leq K \tag{1.1}$$

où x^j est la $j$ème coordonnée de x, et K la dimension de l'espace de sortie. La fonction d'*activation* ϕ_h de la couche cachée est la fonction logistique

$$\phi_h(z) = \ell(z) = \frac{1}{1+\exp(-z)}.$$

En revanche, ϕ_o peut être logistique ou linéaire ($\phi_o(z) = z$) selon que l'on souhaite faire de la classification ou estimer une régression.

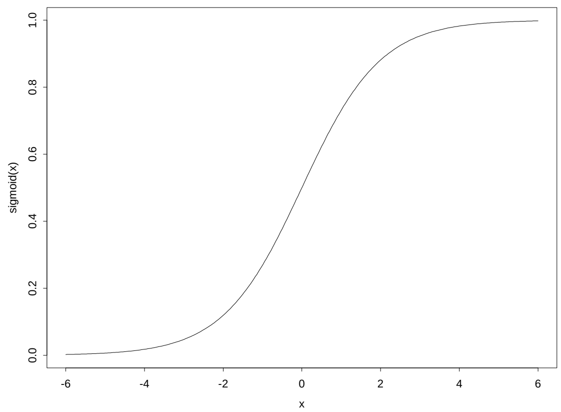

FIG. 1.2 – *La courbe représentatrice de la fonction logistique.*

Comme nous le verrons par la suite (Chapitre 2), un réseau de neurones peut être vu comme un moyen d'approcher une application non-linéaire prise dans un ensemble assez général. Par exemple, une fonction continue f peut être approchée uniformément sur tout compact par un réseau de neurones en augmentant la taille de la couche cachée (et le nombre de paramètres).

Malheureusement, les démonstrations de tels résultats ne sont pas "constructives". En pratique, les poids et les biais doivent être ajustés en minimisant un critère d'erreur. Un choix possible est le critère des moindres carrés, fondé sur un échantillon (x_i, y_i) de taille n

$$E(\theta) = \sum_{i=1}^{n} \|g_i(\theta) - y_i\|^2 = \sum_{i=1}^{n} E_i(\theta).$$

Dans cette équation, y_i est la sortie correspondant à l'exemple x_i et on pose

$$g_i = (g^k(x_i, \theta))_{k=1,\dots,K}.$$

D'autres critères ont été proposés. L'alternative la plus fréquente est l'*entropie*. Mais, on l'utilise en général lorsque les données modélisées sont des probabilités (dans ce cas, $y^\ell, g^\ell \in [0,1]$). Ce critère revient à minimiser

$$E(\theta) = \sum_i \sum_\ell \left[y_i^\ell \log \frac{y_i^\ell}{g_i^\ell} + (1 - y_i^\ell) \log \frac{1 - y_i^\ell}{1 - g_i^\ell} \right] . \tag{1.2}$$

Une manière de s'assurer du bon comportement numérique de l'algorithme de minimisation est d'utiliser la méthode de *régularisation de l'erreur* qui consiste à remplacer E par

$$E(\theta) + \lambda \sum_{ij} \theta_{ij}^2 , \quad \lambda > 0,$$

appelée *weight decay*, spécifique aux réseaux de neurones. L'ajustement du paramètre θ se fait en utilisant une méthode numérique traditionnelle. Les premières implantations étaient fondées sur la méthode du gradient. Toutefois, les méthodes de second ordre, en particulier les méthodes de quasi-Newton, sont largement plus efficaces.

Exercice 1.2.1 *Calculer les dérivées partielles de la fonction E par rapport aux paramètres θ_{ij}. Écrire l'algorithme de minimisation utilisant la méthode du gradient à pas constant. Cet algorithme s'appelle rétro-propagation de l'erreur. Comprenez-vous pourquoi ?*

1.2.1 Le programme nnet

Un logiciel pour ajuster des modèles de réseaux de neurones est disponible en S-Plus (`nnet`). Ce logiciel traite le problème de minimisation du critère d'erreur comme un problème d'optimisation, selon la méthode quasi-Newton appelée BFGS. Les dérivées premières sont calculées selon la technique de rétro-propagation. La fonction première de l'algorithme `nnet` est donc le calcul d'un paramètre $\hat{\theta}$ correspondant à un minimum (local) de l'erreur $E(\theta)$. Le format d'appel sous S-Plus est le suivant.

```
> modele <- nnet(x, y, size, Wts, linout=F, entropy=F, softmax=F, skip=F,
rang=0.7, decay=0, maxit=100, trace=T)
```

Les paramètres de l'appel sont

`x`	matrice de variables.
`y`	matrice ou vecteur de variables.
`size`	nombre d'unités cachées.
`Wts`	poids initiaux θ_{ij} (par défaut aléatoires !!!).
`linout`	booléen pour une sortie linéaire.
`entropy`	booléen pour entropie plutôt que moindres-carrés.
`softmax`	booléen pour un modèle log-linéaire (optionnel).
`skip`	booléen pour des poids entre entrées et sorties.
`rang`	Si `Wts` est absent, utilise des poids aléatoires `runif(n,-rang, rang)`.
`decay`	paramètre de régularisation λ.
`maxit`	nombre d'iterations maximal effectuées par l'algorithme de Newton.
`trace`	booléen pour la trace de l'optimiseur. Very reassuring!

Le paramètre estimé $\hat{\theta}$ peut être récupéré de la manière suivante

```
> modele$wts
```

On peut utiliser la commande `predict` pour appliquer la fonction ajustée à de nouvelles données. Cette fonction recalcule ainsi l'application $g(x, \hat{\theta})$. Attention, elle attend le format "matrice" en second argument

```
> predict( modele, as.matrix( x.newdata ) ) .
```

La commande `fitted` permet de récupérer les valeurs de la fonction pour les variables d'entrées de l'échantillon

```
> fitted.values( modele).
```

Elle est équivalente à

```
> predict( modele, as.matrix( x ) ) .
```

Essayez aussi `summary`. Cette commande vous indique comment la liste des paramètres estimés est construite.

La fonction `nnet.Hess` permet de calculer le Hessien de l'erreur par rapport au paramètre θ. Cette commande est utile pour s'assurer qu'un minimum local a bien été atteint.

```
> eigen(nnet.Hess( model, x, y), T, only.values=T)$values
```

En général, pour aborder un problème de régression non-linéaire, on utilise l'erreur des moindres carrés, et le paramètre de régularisation peut varier entre 10^{-2} et 10^{-4}. Il est crucial d'utiliser l'option `lin = T` car les sorties ne sont pas nécessairement dans l'intervalle $(0, 1)$. Il y a autant de cellules d'entrée et de cellules de sortie que de dimensions des variables correspondantes.

En général, pour aborder un problème de classification, on utilise l'erreur d'entropie (log-vraisemblance) et le paramètre de régularisation peut varier entre 10^{-1} et 10^{-3}. Il est crucial d'utiliser l'option `lin = F` car les sorties sont nécessairement dans l'intervalle $(0, 1)$. En effet, comme on le verra au chapitre 3, on ne modélise pas le numéro de la classe mais plutôt la probabilité (conditionnelle) d'être dans une classe. Supposons qu'il y ait K classes. Il y aura autant de cellules de sortie. Pour utiliser `nnet`, il est nécessaire de transformer les étiquettes de classe en vecteurs unitaires. Dans ce cas, la classe

$$c_i \in \{1, \dots, K\}$$

correspond à la sortie

$$y_i = (e_1, \dots, e_K)$$

où $e_k = 1$ si $c_i = k$ et $e_k = 0$ sinon.

1.3 TP1 : Jouons avec `nnet`

La première séance de TP est une séance de familiarisation avec la commande `nnet`. Il s'agit d'explorer le comportement du logiciel sur un exemple de régression unidimensionnelle. On programmera aussi le calcul des dérivées partielles de l'erreur (moindres-carrés, entropie) selon la méthode de rétro-propagation.

1.3.1 L'environnement S

L'un des objectifs de ce cours est de se familiariser en général avec le langage S. L'aide en ligne et les séances d'introduction présentées lors du stage de programmation scientifique seront très utiles. Le manuscript de Venables et Smith (1992) nous servira de documentation pour l'utilisation du langage S. Les individus intéressés pourront aussi consulter le site électronique Statlib (Carnagie Mellon University) qui contient de nombreuses bibliothèques (dont `nnet`) d'accès libre pour ce logiciel.

1.3.2 Un exemple à explorer

Nous supposons que les variables d'entrée et de sortie sont unidimensionnelles et liées par la relation suivante. Pour tout i de 1 à n,

$$y_i = g(x_i) + \epsilon_i \tag{1.3}$$

où g est une fonction inconnue et ϵ_i un bruit centré, que l'on supposera être de loi normale $\mathcal{N}(0, \sigma^2)$. On cherche à estimer la fonction inconnue à l'aide d'une fonction réseau de neurones $g(x;\theta)$, comportant une unique unité d'entrée et une unique unité de sortie. Nous utilisons le critère d'erreur quadratique. Notre exemple de référence est issu de (Wahba et Wold, 1975). Il s'agit de la fonction

$$g(x) = 4.26(e^{-x} - 4e^{-2x} + 3e^{-3x}) \ , \quad x \in [0, 3.0] \ .$$

Les x_i sont choisis selon un *plan d'expérience* régulier

$$x_i = 3i/n \ , \quad 1 \leq i \leq n = 200$$

et $\sigma = 0.2$ (Figure 1.3).

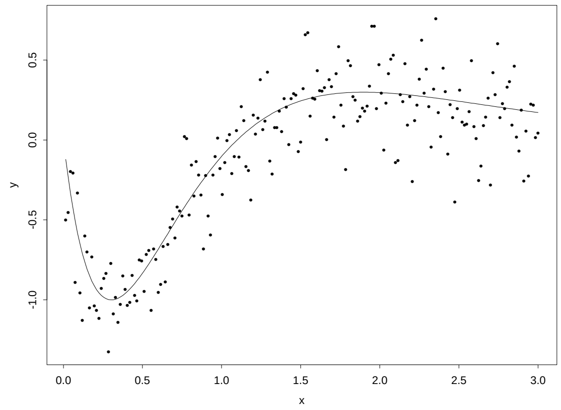

FIG. 1.3 – *La courbe de Wahba et Wold ainsi qu'un échantillon de 200 données simulées (*$\sigma = 0.2$*)*

1) Créer un échantillon selon le modèle 1.3. Tracer le graphe de la fonction g.
2) À l'aide de la commande `scan`, saisir sous S-Plus, les données contenues dans les fichiers x1 et y1 du répertoire Nnet.
3) Ajuster un modèle de réseau de neurones pour estimer g d'une part, et la régression de y_1 par x_1 d'autre part
   – en faisant varier la dimension du réseau de neurones :

```
> set.seed(12)
> mod _ nnet( x,y, size = 20, lin = T, maxit = 1000, trace = F)
```

```
> lines( x, fitted(mod))
> mod <- nnet( x,y, size = 1, lin = T, maxit = 1000, trace = F)
> lines( x, fitted(mod))
```

– en initialisant les poids de manières différentes,

```
> set.seed(1)
> mod _ nnet( x,y, size = 20, lin = T, maxit = 1000, trace = F)
> lines( x, fitted(mod))
> mod <- nnet( x,y, size = 20, lin = T, maxit = 1000, trace = F)
> lines( x, fitted(mod))
```

– en modifiant le test d'arrêt de l'algorithme d'optimisation,

```
> mod <- nnet( x,y, size = 20, lin = T, maxit = 100, trace = F)
```

– en modifiant la valeur de `decay`,

```
> mod <- nnet( x,y, size = 20, lin = T, maxit = 1000,
decay = 0.001, trace = F)
```

– en variant la taille de l'échantillon ou sa variance.

La commande `set.seed` permet d'initialiser le générateur aléatoire de S-Plus. Pour le jeu de données (x_1, y_1), un réseau de taille 45 vous parait-il sur-dimensionné ? (et 25) ? Chercher un exemple pour lequel l'ajustement parait visuellement meilleur avec une taille 45.

4) Comparer les ajustements obtenus avec et sans le paramètre de régularisation. Conclusion ?

5) Utiliser les commandes `predict` et `fitted` pour recalculer l'erreur d'ajustement $E(\hat{\theta})$ à partir de l'échantillon et des paramètres estimés. Si l'objet créé par `nnet` s'appelle `modele`, alors le vecteur des poids estimés est `modele$wts`. L'appel `modele$n` renvoie les dimensions du réseau. Il est utile de comprendre comment le vecteur des poids correspond à l'architecture du réseau de neurones. On obtient ce renseignement grâce à l'appel de `summary`. La cellule est 0 est une cellule d'entrée artificielle, fixée à la valeur $+1$ et destinée à coder les (α_i). La cellule d'entrée est numérotée 1, les cellules cachées sont numérotées de 2 à `size` $+1$ et la cellule de sortie est `size` $+2$. Au total, il y a 3 `size` $+1$ paramètres. Voici un exemple de recalcul de la fonction "réseau de neurones" pour les exemples de regression unidimensionnelle

```
reseau.f <- function(mod,x)
{
w <- mod$wts
size <- mod$n[2]

w.ih <- w[seq(2,2*size, length=size)]
alpha.h <-  w[seq(1,2*size-1,length=size)]
w.ho <- w[(2*size + 2):length(w)]
```

```
return(w[ 2*size + 1] + sum( w.ho*logist(alpha.h + w.ih*x )))
}
```

6) Pour un réseau de taille 20, estimer par simulation de 30 échantillons (y_i) différents l'erreur

$$E[(g(x;\hat{\theta}) - g(x))^2]$$

le biais

$$\text{biais}(g(x;\hat{\theta})) = E[g(x;\hat{\theta})] - g(x)$$

et la variance

$$\text{Var}(g(x;\hat{\theta})) = E[g(x;\hat{\theta})^2] - E[g(x;\hat{\theta})]^2$$

de l'estimateur $g(x;\hat{\theta})$, pour une valeur $\lambda = 0.1$ et pour une valeur $\lambda = 0.0001$. On prendra garde à l'initialisation par défaut des poids qui est aléatoire. Afin de minimiser son effet, on pourra initialiser les poids à partir de ceux obtenus pour un modèle de référence construit à partir d'un échantillon arbitraire.

Que suggèrent les résultats ? Prendre des échantillons de taille 100 (de taille 1000 !), augmenter σ^2. Que se passe-t-il ?

7) Créer une fonction de S-Plus qui recalcule les dérivées partielles de l'erreur E par rapport aux paramètres du modèle, prises au point $\hat{\theta}$ estimé par `nnet`.

1.4 TP2 : Visualisons des classes

Dans ce second TP, il s'agit de classification à partir de données réelles et simulées.

1.4.1 Un exemple artificiel

L'objectif de cette section est de créer un exemple artificiel comportant 100 données et deux classes. Les données seront uniformément réparties dans le carré $[-1.25, +1.25] \times [-1.25, +1.25]$. La première classe, d'étiquette “i”, correspondra à l'intérieur du disque unité et la seconde, d'étiquette “o”, correspondra à son extérieur. Visualisez les données, et sauvez les sous forme de table[2] dans un fichier que l'on nommera “artificiel.dat”. Les commandes `sapply` ou `apply` sont très commodes pour convertir des vecteurs, des matrices ou des listes (de booléens en caratères par exemple).

Commandes S-Plus: plot, apply (ou sapply), runif, text, data.frame, write.table

2. Attention ! Dans un tableau de données data.frame les variables qualitatives ”o” et ”i” seront aussi vues comme les niveaux d'un facteur et numérotées implicitement 1 et 2

1.4.2 Le fichier *crabs.dat*

Le fichier *crabs.dat* est un fichier de données réelles, recueillies par le biologiste Mahon (1974), à partir de mesures effectuées sur 200 crabes de la variété *Leptograpsus variegatus*, sur la côte ouest de l'Australie. Les crabes sont bleus ou oranges. Mahon a identifié 50 mâles et 50 femelles de chaque couleur. Pour chaque specimen, il a effectué 5 mesures

- la longueur de la carapace (CL);
- la largeur de la carapace (CW);
- la taille du lobe frontal (FL);
- la largeur de l'arrière du crabe (RW);
- l'épaisseur du corps (BD).

On peut lire ce fichier sous S-Plus, à l'aide de la commande

```
> crabs <-  read.table("crabs.dat", header=T, as.is=T)
> attach(crabs)
```

Le fichier *crabs.dat* est entré sous le format `data.frame`.

Nous souhaitons construire une règle pour prédire le sexe d'un crabe, à partir de deux mesures seulement : FL et RW, par exemple. Pour cela nous utilisons un réseau de neurones pour modéliser les probabilités conditionnelles pour que le crabe soit un male ou une femelle connaissant les variables FL et RW[3] (en principe, la somme des deux probabilités est égale à 1). Le critère que l'on choisit consiste à minimiser l'entropie et les unités de sorties seront "logistiques". On définira donc un modèle de la manière suivante

```
> set.seed(1)
> model <- nnet(dcrabs, Sex.type, size = 6, decay = 0.01, entropy = T,
 maxit = 500)
```

où `dcrabs` est l'échantillon des mesures des paramètres FL et RW (en fait des logarithmes), `Sex.type` est une liste de vecteurs binaires de la forme $(1,0)$ ou $(0,1)$ selon le sexe du crabe. La commande `predict` permet de calculer les valeurs estimées pour un couple quelconque (x_{fl}, x_{rw}) selon la syntaxe

```
> predict( model , cbind( xfl , xrw) )
```

Attention, `predict` attend une matrice en second argument. Pour cette raison, on utilise `as.matrix` ou `cbind`.

La comparaison des probabilités conditionnelles fournies par cette commande permet de prédire le sexe du (nouveau) spécimen mesuré. Application : déterminer le sexe d'un spécimen dont les mesures logarithmiques sont $(2.6, 2.4)$. Visualiser les données avec leurs étiquettes de classe, comme dans le paragraphe précédent (et la figure 1.4).

Commandes S-Plus : read.table, sapply, nnet, predict, which.is.max.

3. Il est possible que cet exemple m'attire des ennuis.

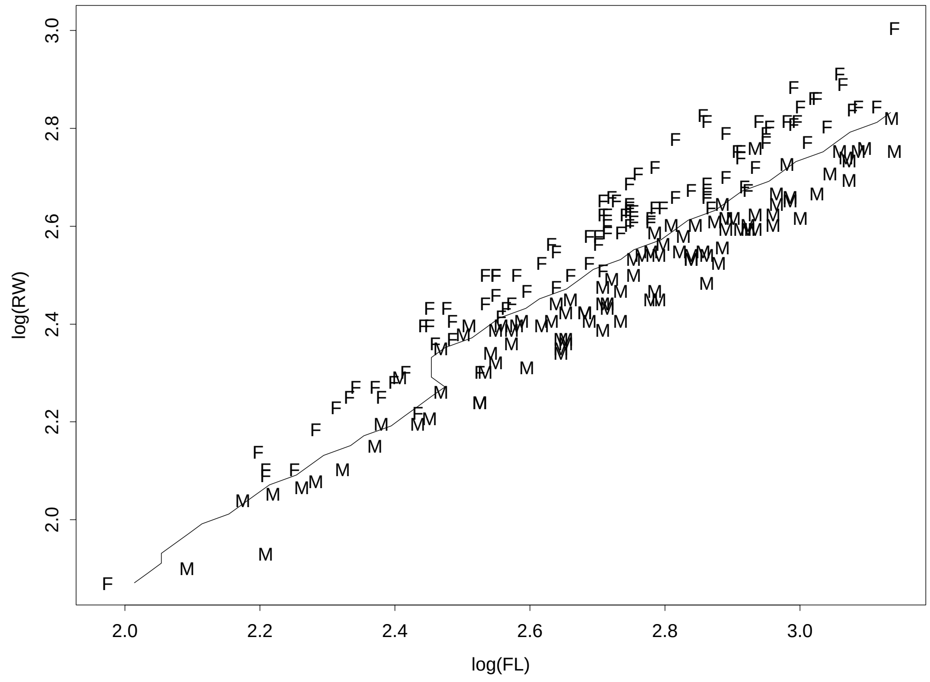

FIG. 1.4 – *Données associées aux crabes leptograpsus. Discrimination entre males (M) et femelles (F) selon la taille du lobe frontal et la largeur de l'arrière de la carapace.*

1.4.3 Le fichier *Cushings.dat*

Le fichier *Cushings.dat* est un fichier de données biomédicales (Aitchison et Dunsmore, 1975), associées à un syndrôme de dysfonctionnement de la glande adrénale: le syndrôme de *Cushings*. Pour ce syndrôme, trois types de diagnostic peuvent être effectués. Ils sont représentés par les symboles a, b, c (sans rentrer dans les détails, notons que a signifie "adénomie", b signifie "hyperplasie bilatérale", c "carcimonie"). Ces symboles représentent la cause du dysfonctionnement : une sécrétion trop importante de cortisol. Les données observées sont des mesures de stéroides métabolites obtenues à partir d'analyses d'urine.

Visualiser les données avec leurs étiquettes de classe, comme dans la figure 1.5.

1.4.4 Visualisation de la prédiction

L'objectif de cette séance de travaux pratiques est de visualiser les frontières séparant les différentes classes pour une prédiction effectuée à l'aide d'un réseau de neurones. Il s'agit de tracer ces frontières pour les fichiers présentés lors des sections précédentes.

Pour cela, une méthode consiste à discrétiser la variable d'entrée, puis à prédire le type de cette variable pour chaque point de la discrétisation. Disons que la discrétisation comporte 50×50 points. Soit K le nombre de classes. On peut remplir une matrice

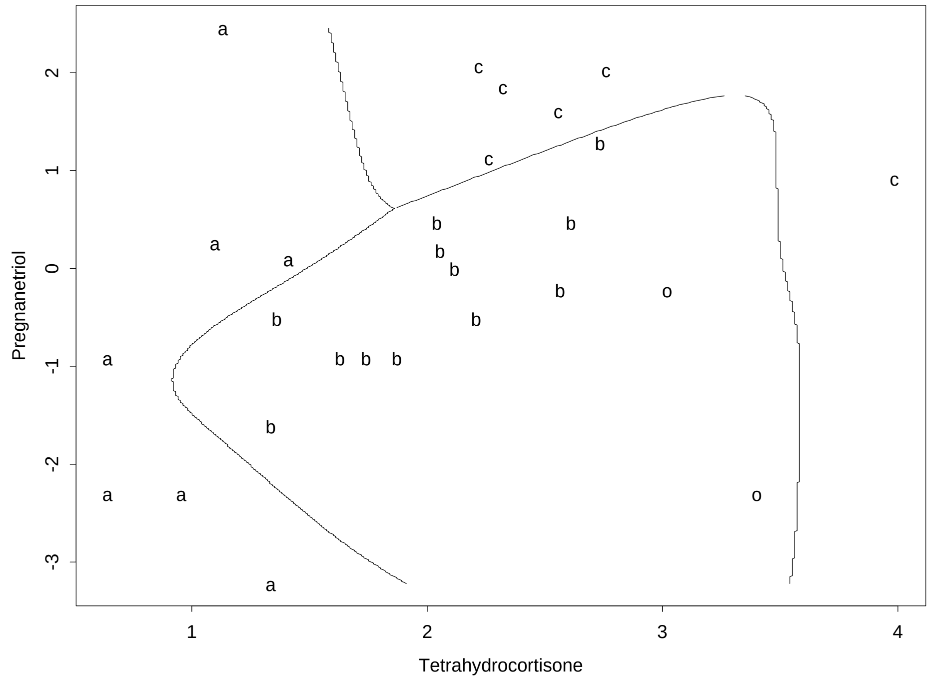

FIG. 1.5 – *Les données du syndrôme de Cushings. Trois types de diagnostics a, b ou c correspondent aux différentes mesures de tétrahydrocortisone et prégnanétriol. Les frontières entre classes ont été estimées à l'aide d'un réseau de neurones.*

M de dimension $50 \times K$, que l'on initialise avec la valeur NA. Puis on prédit ligne après ligne. Pour la ligne i, on mémorise l'abscisse du premier point de sortie de la classe ℓ dans $m_{i\ell}$. Cette méthode est toutefois imparfaite dans le cas de frontières plus complexes que celles qui existent dans les problèmes étudiés. La commande `lines` permet de tracer les frontières, car elle ne prend pas en compte les valeurs NA. La commande `range` peut aussi être utile pour fixer la zone à tester.

Chapitre 2

Régression non-linéaire

2.1 Présentation

Dans tout ce chapitre, nous considèrons que la variable de sortie y est de dimension 1 afin d'alléger les notations et par souci de clarté. Nous supposons que les variables d'entrée et de sortie sont liées par la relation suivante. Pour tout i de 1 à n,

$$y_i = g(x_i) + \epsilon_i \tag{2.1}$$

où g est une fonction inconnue et ϵ_i un bruit centré, que l'on supposera être de loi normale $\mathcal{N}(0, \sigma^2)$. On cherche à estimer la fonction inconnue à l'aide d'une fonction réseau de neurones $g(x; \theta)$ en utilisant le critère d'erreur quadratique. Pour un réseau de neurones, le paramètre θ représente l'ensemble des poids w_{ij} et α_i.

Les résultats énoncés dans ce chapitre s'applique majoritairement à la problèmatique de classification binaire présentée rapidement dans le chapitre introductif. On parle alors plus volontier de régression logistique. Dans ce cas, le réseau de neurones sert à estimer la probabilité conditionnelle

$$g(x) = P(Y = 1 | X = x) \ .$$

Nous avons choisi cependant de séparer la discussion concernant les problèmes de régression de celle concernant les problèmes de classification. Nous y consacrerons le chapitre suivant.

Supposons que nous ne disposons que d'une seule observation, notée x, que nous considérons comme la réalisation de la variable aléatoire X. Pour alléger les notations, nous noterons l'espérance conditionnelle de Y sachant $X = x$ de la manière suivante

$$E[Y|\ X = x] = E[Y|\ x] \ .$$

Nous cherchons la fonction $g(x)$ qui minimise l'erreur de prédiction que l'on peut faire sur Y sachant que $X = x$

$$g(x) = \arg\ \min_f E[(Y - f(x))^2|\ x] \ .$$

Exercice 2.1.1 *Montrez, à condition que les espérances soient bien définies, que l'espérance conditionnelle (ou régression) est une bonne solution. (Consulter éventuellement le cours de première année.)*

Indication: Pour toute fonction f (x fixé), nous avons

$$E[(Y - f(x))^2 |\, x] = E[(Y - E[Y|\, x] + E[Y|\, x] - f(x))^2 |\, x]$$

Développer le membre de droite pour montrer que

$$E[(Y - f(x))^2 |\, x] \geq E[(Y - E[Y|\, x])^2 |\, x].$$

■

La minimisation en θ du critère empirique

$$E(\theta) = \sum_{i=1}^{n} (y_i - g(x_i; \theta))^2$$

ou de manière équivalente de $E(\theta)/n$ constitue un moyen d'estimer la régression $E[Y|\, x]$ sous la forme $g(x, \hat{\theta})$ (pourquoi ?).

2.2 Décomposition biais/variance

Le problème d'estimation de la régression est donc de “reconstruire” une fonction de régression $g(x)$ à partir d'un ensemble d'apprentissage constitué de données conditionnellement indépendantes

$$\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\} .$$

L'idée généralement développée est d'utiliser un modèle paramétrique de la forme $g(x; \theta)$. Notons que la dimension de θ n'est pas connue à d'avance, et peut être arbitrairement grande dans le but d'approcher n'importe quelle fonction $g(x)$. Dans cette optique, on parle en fait d'estimation non-paramétrique (le véritable paramètre est de dimension infinie). Étant donné un ensemble $\mathcal{D}$, notons $\hat{\theta}$ le paramètre estimé en utilisant la procédure des moindres carrés. Par suite, l'estimation de la régression sera $g(x; \hat{\theta})$. Une mesure naturelle de l'efficacité de $g(x; \hat{\theta})$ comme prédicteur de Y est

$$E[(Y - g(x; \hat{\theta}))^2 |\, x; \mathcal{D}] .$$

En utilisant le même subterfuge que dans l'exercice 2.1.1, montrez (exercice) qu'il est tout aussi naturel de choisir la grandeur

$$(g(x; \hat{\theta}) - E[Y|\, x])^2.$$

Maintenant, prenons en compte le fait que l'ensemble d'apprentissage $\mathcal{D}$ est un ensemble particulier aléatoire, dont dépend fortement $\hat{\theta}$. Nous obtenons une nouvelle mesure d'erreur en moyennant sur tous les ensembles $\mathcal{D}$ (de taille n fixée)

$$E[E[(g(x;\hat{\theta}) - E[Y|\ x])^2|\ \mathcal{D}]] = E[(g(x;\hat{\theta}) - E[Y|\ x])^2].$$

Il se peut que, pour un ensemble $\mathcal{D}$ particulier, $g(x;\hat{\theta})$ soit une excellente approximation de la régression (et ainsi un prédicteur presque optimal de Y). Mais en général, $g(x;\hat{\theta})$ varie substantiellement avec $\mathcal{D}$. Il se peut aussi qu'en moyenne $g(x;\hat{\theta})$ soit assez éloigné de la véritable fonction. L'espérance sera alors grande et la prédiction peu fiable. Un moyen de comprendre ces deux différentes sources d'erreur d'estimation est la décomposition biais/variance.

Proposition 2.2.1 *Nous avons, en supposant* (X, Y) *indépendant de* $\mathcal{D}$,

$$E[(g(x;\hat{\theta}) - E[Y|\ x])^2|\ x] = \mathit{biais}(g(x;\hat{\theta}))^2 + \mathit{Var}(g(x;\hat{\theta}))$$

où

$$\mathit{biais}(g(x;\hat{\theta})) = E[g(x;\hat{\theta})] - E[Y|\ x]$$

et

$$\mathit{Var}(g(x;\hat{\theta})) = E[\big(g(x;\hat{\theta}) - E[g(x;\hat{\theta})]\big)^2]\ .$$

Démonstration. Exercice Peu Difficile. ■

Si, en moyenne sur $\mathcal{D}$, $g(x;\hat{\theta})$ est différent de $E[Y|\ x]$, alors l'estimateur est biaisé. Bien entendu le biais est une source d'erreur. Il correspond à l'inexactitude du modèle. Toutefois un estimateur sans biais peut aussi avoir de faibles performances en moyenne, s'il est très sensible aux variations de l'ensemble $\mathcal{D}$ (voir la figure 2.1).

En général, il y a un compromis à trouver entre la minimisation du biais et de la variance que Grenander (1951) appelle *principe d'incertitude*. Typiquement, un modèle comportant peu de paramètres peut être très faux et le biais sera grand. En revanche, ce modèle sera moins sensible aux variations des données qu'un modèle comportant beaucoup de paramètres. Le second modèle sera toutefois certainement plus précis en moyenne.

Ces remarques suggèrent que la minimisation du biais et de la variance sont deux objectifs antagonistes. Le biais diminue lorsque le nombre de paramètres du modèle augmente. Mais alors, la variance augmente aussi. Cela suggère de pénaliser l'erreur $E(\theta)$ par addition d'un terme croissant en fonction du nombre de paramètres. Cette idée sera exploitée largement lors de la section suivante.

2.3 Sélection de modèle

Dans cette section, nous nous demandons comment déterminer la complexité d'un modèle à l'intérieur d'une famille paramétrique $g(x;\theta)$. Par exemple, combien de cellules

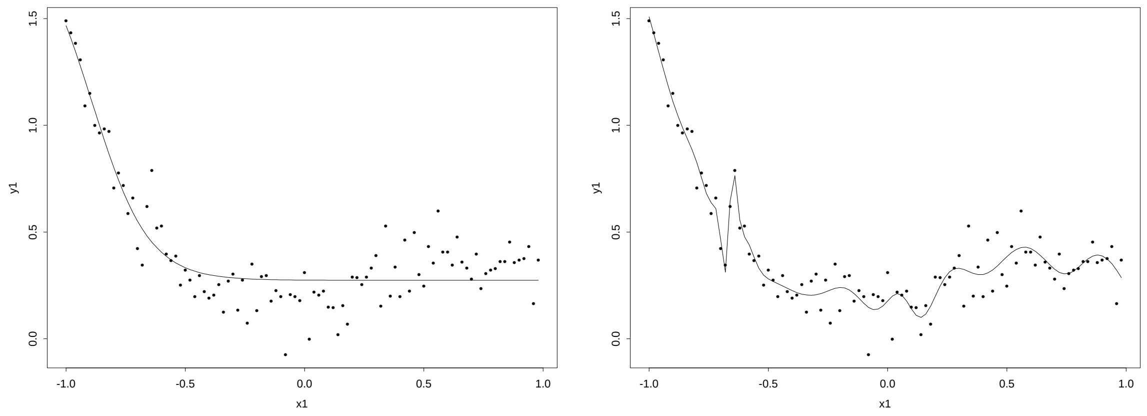

FIG. 2.1 – *Le biais et la variance. Le biais apparait à gauche pour un estimateur peu paramétré (1 neurone) peu sensible à l'échantillon et la variance à droite pour un estimateur surparamétré (45 neurones).*

cachées sont nécessaires dans un réseau de neurones pour un problème de régression non-linéaire. Trois approches différentes sont traditionnellement adoptées en statistique pour résoudre ce problème.

La première approche s'appelle **sélection itérative**. Souvent, l'itération a pour effet de simplifier le modèle. On part d'un modèle surparamétré, puis l'on considère une suite de sous-modèles imbriqués que l'on compare successivement grâce à des tests utilisant le rapport de vraisemblance. Cette méthode est traditionnellement utilisée en régression linéaire. Sous S-Plus, la commande *anova* permet d'implanter la procédure.

La seconde méthode s'appelle **pénalisation de l'erreur** ou de la log-vraisemblance. Nous avons vu dans la section précédente que des modèles comportant trop peu de paramètres estiment mal la régression car ils sont biaisés. D'autres part les modèles comportant beaucoup de paramètres sont très variables. Il y a donc un compromis à trouver en pénalisant l'erreur (qui diminue avec la complexité du modèle) par un terme qui augmente avec l'ordre du modèle. C'est l'esprit des critères AIC et NIC que nous présenterons dans un paragraphe suivant.

La troisième méthode est pûrement calculatoire : il s'agit de la méthode de **validation croisée**. Dans cette méthode, on utilise une partie des données pour l'estimation du paramètre θ et la partie complémentaire, que l'on appelle *sous-ensemble de validation*, pour évaluer l'erreur de prédiction. Pour obtenir une évaluation raisonnable (pas trop variable), on moyenne la prédiction en choisissant un nombre important de sous-ensembles de validation.

2.3.1 Régularisation

La régularisation est un préalable essentiel à la sélection de modèle, destinée à minimiser l'importance des pièges numériques auxquels conduit inévitablement la dynamique de minimisation de l'erreur.

La minimisation de l'erreur se fonde sur l'algorithme de rétropropagation de l'erreur. Malheureusement, la procédure numérique ne converge pas nécessairement vers le paramètre optimal (le maximum de vraisemblance). En effet, il existe des minima locaux de l'erreur, et surtout des plateaux, qui peuvent être des pièges redoutables pour l'estimation. Cette dernière est en fait sensible à l'initialisation de l'algorithme numérique. A cause de ces pièges, un modèle peut paraître meilleur qu'un autre alors qu'en réalité il ne l'est pas. La figure 2.2 illustre ce paradoxe. Dans la figure de gauche, un réseau de 45 neurones cachés (136 paramètres) conduit à une estimation qui parait visuellement satisfaisante. Toutefois, le modèle est clairement surparamétré (il n'y a que 100 données). Dans la figure de droite, un modèle à 10 neurones est utilisé (31 paramètres) et l'estimateur semble en fait plus sensible au jeu de données. Cela est en contradiction avec la décomposition biais/variance (le résultat devrait être inversé). Cet effet s'explique de la manière suivante

- l'optimiseur est local, et il peut être piégé (dans le cas de gauche) dans une solution qui semble "visuellement" satisfaisante, bien que le modèle soit certainement mauvais,
- l'arrêt prématuré (à 200 itérations) joue un rôle de régularisation en évitant une minimisation trop fine, et augmente le biais au détriment de la variance.

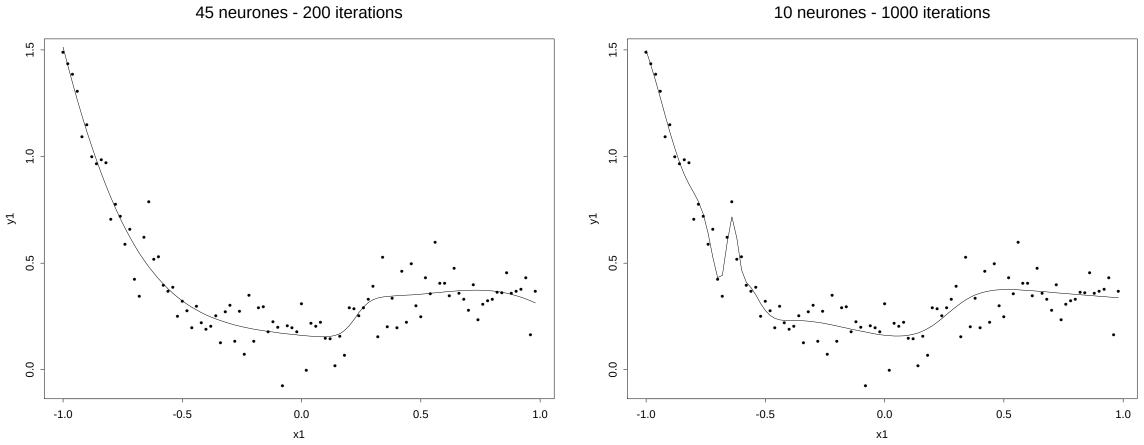

FIG. 2.2 – *Une situation paradoxale par rapport à décomposition biais/variance. Le modèle surparamétré semble visuellement meilleur (à gauche). En fait, l'arrêt prématuré à 200 itérations a pour effet de diminuer la variance (au prix d'un biais certainement un peu plus élevé).*

La présence de plateaux est essentiellement dûe à la saturation des unités logistiques lorsque les paramètres θ_{ij} associés deviennent grands en valeur absolue. Une manière d'éviter ce phénomème de saturation est de pénaliser les valeurs élevées des paramètres. Cette méthode dûe à Hinton (1986) s'appelle *weigth decay* et consiste à remplacer l'erreur E par

$$E(\theta) + \lambda C(\theta) = E + \lambda \sum_{ij} \theta_{ij}^2$$

où $\lambda > 0$ est un paramètre de régularisation que l'on aura à estimer séparément des θ_{ij}. La méthode recommandée pour fixer λ est la méthode de validation croisée, que nous aborderons dans la suite et par la méthode de dichotomie.

Commentaires. Dans cette démarche de régularisation, la minimisation de l'erreur des moindres carrés se fait sous la contrainte de minimiser aussi l'erreur $C(\theta)$. Le paramètre *decay* n'est rien d'autre qu'un lagrangien à fixer convenablement. Notons que cette démarche opère une sélection implicite des cellules cachées. En effet, contraindre un poids afférant à une cellule cachée à être nul revient à tuer cette cellule. Ainsi, *decay* peut s'utiliser à partir de modèles qui semble à priori surparamétrés, c'est à dire trop variables. Le rôle de *decay* consistera à diminuer la variance de l'estimateur, au détriment d'un biais plus élevé mais que l'on espère raisonnable.

Commentaires. Notons, comme le montre la figure précédente qu'il est aussi possible de régulariser par arrêt prématuré de l'optimiseur (*early stopping*). Bien que des travaux récents confirment mathématiquement l'intérêt de cette procédure, elle reste difficile à utiliser.

Exercice 2.3.1 *(Peu Difficile) Justifier l'existence de plateaux à partir du calcul de la dérivée de la fonction logistique.*

La figure 2.3 montre l'effet de régularisation sur l'estimation de la régression. Dans l'estimation en pointillé, l'algorithme numérique d'optimisation est lancé sans régulariser ($\lambda = 0.0$) à partir d'un réseau à 10 cellules cachées (certainement trop!). L'estimation est renouvellée, mais en régularisant à l'aide d'un paramètre $\lambda = 0.001$. L'effet de localisation à disparu, et l'estimation apparait nettement plus "régulière". En résumé, les avantages de *decay* sont la baisse de sensibilité à l'initialisation des poids (`Wts`) et la sélection implicite des cellules cachées (3 neurones produiraient le même estimateur). La figure 2.3 suggère que le biais reste très faible.

2.3.2 Critère d'information et pénalisation

En statistique, la modélisation vise deux objectifs qui peuvent être dissociés : **expliquer** et **prévoir**. Dans la première démarche, on cherche à savoir, par exemple, quelles sont les variables qui influent de manière significative sur un phénomène (de telles variables sont par ailleurs appelées *explicatives*). L'estimation est alors une démarche cruciale et doit être précise. Dans la seconde démarche, on cherche un modèle qui fait le moins d'erreur possible (disons en moyenne) sur de nouvelles données qui n'ont pas servi pour l'estimation (ou l'apprentissage). Il n'est alors pas forcément utile de minimiser parfaitement le critère d'estimation (cela explique le succès des méthodes de régularisation qui modifient le critère, ou de l'arrêt prématuré de l'algorithme d'optimisation).

Pour un réseau de neurones, l'objectif recherché est majoritairement la prédiction (toutefois, la sélection de modèle permet aussi de réaliser le premier objectif si on le

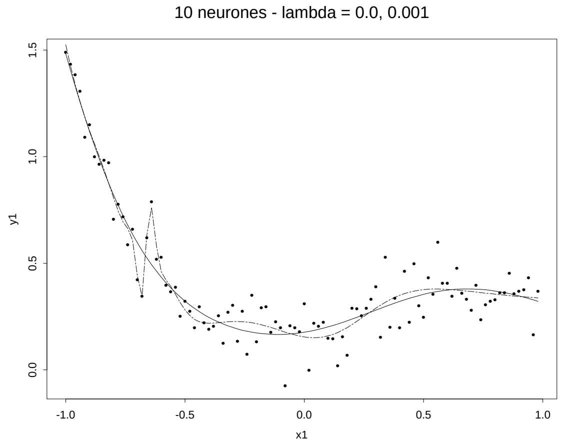

FIG. 2.3 – *L'effet (spectaculaire) de régularisation. La courbe en pointillé est obtenue à partir d'un réseau de neurones à 10 cellules cachées (31 paramètres) et $\lambda = 0.0$. La courbe pleine est obtenue à partir du même modèle en utilisant $\lambda = 10^{-3}$.*

souhaite). La sélection de modèle se fonde sur l'erreur de prédiction que nous allons essayer d'estimer grâce aux propriétés asymptotiques du maximum de vraisemblance (MV). Nous énonçons les propriétés en question dans un cadre statistique général, puis nous précisons l'application aux réseaux de neurones.

2.3.3 Propriétes asymptotiques du MV

La situation la plus courante en statistique est que l'on dispose d'un échantillon

$$y_1, \ldots, y_n$$

de loi inconnue, de densité $p(y)$. Modéliser revient à imposer une structure statistique, que nous supposons paramétrique, de densité $p(y;\theta)$. On commettra donc une erreur de modélisation, dûe au fait que la vraie densité $p(y)$ n'appartient pas forcément à la famille $p(y;\theta)$.

Définition 2.3.1 *L'information de Kullback-Lieber est la grandeur définie par*

$$I(\theta) = -E[\log(p(Y;\theta)/p(Y))] \ ,$$

où Y est une variable aléatoire de densité p.

Commentaires. L'information mesure intuitivement la distance qui sépare la loi de densité $p(y;\theta)$ de la loi de densité $p(y)$. Il ne s'agit toutefois pas exactement d'une distance entre les deux densités : la grandeur n'est pas symétrique.

En fait, la grandeur $I(\theta)$ mesure l'erreur relative moyenne que l'on commet en cherchant à prédire la variable Y à l'aide de la densité $p(y;\theta)$. Formellement, on aura intérêt à minimiser cette erreur. C'est à dire à chercher

$$\theta_0 = \arg\ \min I(\theta)\ .$$

Cela revient à rechercher

$$\theta_0 = \arg\ \min \mathcal{E}_{\mathrm{pr}}(\theta)$$

où $\mathcal{E}_{\mathrm{pr}}$ est l'*erreur de prédiction* définie comme suit :

$$\mathcal{E}_{\mathrm{pr}}(\theta) = -E[\log(p(Y;\theta))]\ .$$

On supposera par la suite que θ_0 est unique.

Dans un problème de régression défini par l'équation (2.1), utiliser le critère des moindres carrés revient à chercher un estimateur de maximum de vraisemblance conditionnel aux x_i.

Exercice 2.3.2 *(Peu Difficile) Établir la relation entre l'erreur quadratique et la log-vraisemblance conditionnelle associée à la famille $g(x,\theta)$.*

Démonstration. (Indication) Elle est liée linéairement à l'erreur des moindres carrés régularisée

$$L(\theta) = -\frac{E(\theta)}{2\sigma^2} - n\log(\sqrt{2\pi}\sigma).$$

■

Exercice 2.3.3 *(Facile) Supposons que $p(y;\theta)$ est la densité de la loi $\mathcal{N}(\theta,1)$, et par exemple, que $p(y)$ est la densité de la loi $\mathcal{N}(0,2)$. Calculer l'erreur de prédiction pour $\theta = 1$. Quelle est la valeur de θ_0 ?*

Puisque $p(y)$ est par hypothèse inconnue, on ne peut malheureusement pas calculer l'erreur de prédiction. On remplace alors cette grandeur par une estimation, que l'on note $E_n(\theta)$

$$\mathcal{E}_n(\theta) = -\frac{1}{n}L_n(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\log p(y_i,\theta)\ .$$

L'estimateur $\mathcal{E}_n(\theta)$ n'est autre, au signe près, que la log-vraisemblance du paramètre θ et l'estimateur de θ_0

$$\hat{\theta}_n = \arg\ \min \mathcal{E}_n(\theta)$$

n'est autre que l'estimateur de *maximum de vraisemblance*. Par la loi forte des grands nombres, $\mathcal{E}_n(\theta)$ est une estimation convergente de $\mathcal{E}_{\mathrm{pr}}(\theta)$. De même, sous des hypothèses raisonnables, on peut montrer que

$$\hat{\theta}_n \to \theta_0$$

presque sûrement. Nous allons préciser le comportement asymptotique de cet estimateur, en montrant que pour l'on peut approcher sa loi par une loi normale, centrée autour de θ_0.

Définition 2.3.2 *Soit J, K les matrices d'information de Fisher*

$$J = -E[\frac{\partial^2}{\partial\theta\partial\theta^T}\log p(Y;\theta_0)]$$

et

$$K = Var(\frac{\partial}{\partial\theta}\log p(Y;\theta_0))$$

où Y est une variable aléatoire de densité $p(y)$.

Commentaires. Le paramètre θ peut être de dimension $d > 1$. Dans ce cas, J est l'espérance de la Hessienne de la fonction $\varphi(\theta) = -\log(p(Y;\theta))$ et K est la matrice de covariance du gradient de cette même fonction. La Hessienne est le gradient sont évalués au point θ_0.

Les matrices d'information J et K permettent d'évaluer l'erreur que l'on commet en remplacant θ_0 par son estimateur $\hat{\theta} = \hat{\theta}_n$.

Proposition 2.3.1 *Nous avons (sous des hypothèses convenables)*

$$\sqrt{n}(\hat{\theta} - \theta_0) \to \mathcal{N}(0, J^{-1}KJ^{-1})$$

en loi.

Démonstration. (Idée de la) L'estimateur $\hat{\theta}$ est solution de

$$L'_n(\hat{\theta}) = \sum_{i=1}^{n} \frac{\partial}{\partial\theta}\log p(Y_i;\theta)|_{\theta=\hat{\theta}} = 0\ .$$

Un développement de Taylor montre que

$$0 = L'_n(\hat{\theta}) = L'_n(\theta_0) + H_n(\theta_1)(\hat{\theta} - \theta_0)$$

où $H_n(\theta_1)$ est la Hessienne de la log-vraisemblance prise en un point θ_1 compris entre $\hat{\theta}$ et θ_0. Nous avons

$$\sqrt{n}(\hat{\theta} - \theta_0) = [-n^{-1}H_n(\theta_1)]^{-1}n^{-1/2}L'_n(\theta_0)\ .$$

En appliquant la loi des grands nombres et le théorème central limite, nous obtenons la convergence en loi

$$\sqrt{n}(\hat{\theta} - \theta_0) \to J^{-1}\mathcal{N}(0, K)\ .$$

Le résultat final provient de la symétrie de J. ■

Ce résultat permet de corriger l'erreur de prédiction faite pour l'estimateur de maximum de vraisemblance, afin de la rendre plus proche de l'erreur minimale absolue.

Proposition 2.3.2 *Sous des hypothèses raisonnables, nous avons*

$$\mathcal{E}pr(\hat{\theta}) \approx \mathcal{E}pr(\theta_0) + \frac{1}{2n} tr(KJ^{-1})$$

où Y est une variable de densité $p(y)$ indépendante de $\hat{\theta}$.

Indication. (Exercice Assez Difficile) On suppose que θ est un paramètre unidimensionnel. Faire un développement de Taylor au voisinage de θ_0 et au second ordre pour évaluer

$$\varphi(\hat{\theta}) = -\log p(y; \hat{\theta}) \ ,$$

puis prendre l'espérance et utiliser la proposition précédente. Généraliser en dimension quelconque. ■

Commentaires. L'estimateur $\hat{\theta}$ est fabriqué à partir de l'échantillon $y_1, \ldots, y_n$. L'erreur de prédiction au point $\hat{\theta}$ représente donc l'erreur moyenne commise sur une donnée n'appartenant pas à l'échantillon

$$\mathcal{E}_{\mathrm{pr}}(\hat{\theta}) = -E[\log(p(Y; \hat{\theta}))] \ .$$

Pour choisir correctement un modèle ajusté par maximum de vraisemblance, on cherchera à minimiser cette erreur. D'autre part, l'erreur minimale

$$\mathcal{E}_{\mathrm{pr}}(\theta_0) = -E[\log(p(Y; \theta_0))]$$

ne peut être calculée que si l'on connait la loi des données p (malheureusement inconnue). Dans la pratique, on l'estimera par l'erreur commise sur l'ensemble des données (en $\hat{\theta}$) : lorsque n est grand, la loi empirique associée à l'échantillon converge vers la loi p (et $\hat{\theta} \approx \theta_0$)

$$\mathcal{E}_{\mathrm{pr}}(\theta_0) \approx E(\hat{\theta}).$$

La proposition 2.3.2 montre qu'il faut corriger cette estimation en ajoutant un terme proportionnel à $\mathrm{tr}(KJ^{-1})$.

Exercice 2.3.4 *(Peu Difficile) Reprendre l'exercice 2.3.3 et calculer J et K. Que se passe-t-il en général lorsque l'on prend $p(y) = p(y; \theta_0)$ pour un θ_0 donné? Montrer que $tr(KJ^{-1})$ correspond exactement à deux fois le nombre de paramètres du modèle.*

2.3.4 Application aux réseaux de neurones

Revenons à l'exemple de la régression

$$Y = g(X) + \epsilon\,, \quad \epsilon \sim \mathcal{N}(0, \sigma^2)\,.$$

On cherche à estimer $g(x)$ à l'aide d'une structure neuronale notée $g(x;\theta)$. Cela sous-entend pour la prédiction que l'on connait x. L'erreur de prédiction s'écrit

$$\mathcal{E}_{\rm pr}(\theta) = -E[\log(p(Y, X;\theta))] = -E[\log(p(Y \,|\, X;\theta))] - E[\log(p_X(X))]\,.$$

Ainsi, minimiser $-E[\log(p(Y \,|\, X;\theta))]$ revient à se replacer dans le contexte du paragraphe précédent. Les erreurs de prédiction et la log-vraisemblance peuvent être remplacée par les erreurs conditionnelles et la log-vraisemblance conditionnelle sachant $X = x$. La densité conditionnelle inconnue est

$$p(y|\ x) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-(y - g(x))^2/2\sigma^2)$$

et le modèle statistique est

$$p(y|\ x;\theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-(y - g(x;\theta))^2/2\sigma^2)\,.$$

Par conséquent, nous avons après intégration, (c'est un exercice)

$$\theta_0 = \mathrm{argmin} E[(Y - g(X;\theta))^2]\,.$$

Commentaires. Les x_i sont souvent déterministes ($x_i = i/n$, par exemple). Pour faciliter la notation, nous avons considéré qu'ils étaient tirés uniformément selon une loi discrète.

Définition 2.3.3 *On note*

$$q^* = tr(K_* J_*^{-1})$$

où

$$J_* = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial^2}{\partial\theta\partial\theta^T}(y_i - g(x_i;\theta))^2|_{\theta=\hat\theta}$$

et

$$K_* = Var((\frac{\partial}{\partial\theta}(y_i - g(x_i;\theta))^2)_i)|_{\theta=\hat\theta}$$

où $Var(v)$ désigne la matrice de covariance empirique du vecteur $(v_i)_{i=1,\ldots,n}$.

En tenant compte du facteur $1/2\sigma^2$, nous pouvons considérer, par la proposition 2.3.2, que $2n\sigma^2\mathcal{E}(\hat\theta)$ peut être estimée par l'erreur commise sur l'ensemble d'apprentissage (à la sortie de *nnet*) plus un terme correctif proportionnel à q^*. Ceci suggère que l'on peut minimiser l'erreur de prédiction dans une classe de modèles en pénalisant le critère d'erreur.

Définition 2.3.4 *Pour un modèle neuronal, on appelle critère NIC (Network Information Criterion) la grandeur*

$$NIC = E(\hat{\theta}) + q^*$$

où $E(\hat{\theta})$ est l'erreur des moindres carrés évaluée au point $\hat{\theta}$. Pour un échantillon de taille n, NIC/n est un estimateur de l'erreur moyenne de prédiction que l'on commettra en utilisant le réseau ajusté en $\hat{\theta}$ pour une donnée indépendante de l'échantillon.

Exercice 2.3.5 *(Obligatoire) Proposer un algorithme de calcul de NIC à partir des commandes de S-Plus pour les problèmes de régression en dimension 1.*

Indication. La seule difficulté (à résoudre) est le calcul de la matrice de covariance du gradient de l'erreur. Ensuite, nous pouvons écrire

```
nic <- function(modele, x,y)
{
        if(class(modele) != "nnet")
                stop("Le modele n'est pas un reseau de neurones")

M <- matrice.derivees(modele, x,y)
K <- var(M)
if (dim(K)[1] != length(modele$wts) )
stop("La dimension de la matrice de covariance
 doit etre egale au nombre de poids")

J <- nnet.Hess( modele, x,y )
J <- solve(J) * length(x)

s1 <- sum( diag( K %*% J) )
s2 <- sum( (y - predict(modele, as.matrix(x)))^2 )

nic <- s2 + s1

return(list(Qstar = s1, Err = s2, NIC = nic))
}
```

Numériquement, S-Plus n'est pas toujours capable d'inverser la matrice hessienne. Dans ce cas, il est bon de faire plusieurs essais. Vérifier à l'aide de données simulées la qualité de l'estimation. ■

Exercice 2.3.6 *(Assez Difficile). Lorsque l'on souhaite utiliser l'erreur régularisée $E+\lambda C$, il est formellement nécessaire de modifier la définition du critère NIC. Pouvez-vous justifier cette modification ? Pourquoi la procédure* `nic` *implante-t-elle aussi le critère régularisé ?*

La sélection de modèle, (c'est à dire le choix du nombre de cellules cachées ou du paramètre λ) se fait par comparaison des valeurs NIC. On choisira le modèle 1 de manière préférentielle au modèle 2 si

$$\mathrm{NIC}_1 < \mathrm{NIC}_2 \, .$$

Commentaires. La sélection de modèle selon le critère NIC ne relève pas d'une démarche statistique complète. Traditionnellement, on choisit plutot de tester une hypothèse nulle, comme par exemple

$$H_0 = \text{la cellule cachée 22 est inutile.}$$

De tels tests, souvent fondés sur la vraisemblance, peuvent être développés pour les réseaux de neurones. Il s'agit essentiellement de tests du χ_2 (White, 1989). Toutefois, ils sont asymptotiques, et souvent très peu réalistes pour des réseaux de neurones. En effet, pour ces derniers, le rapport entre nombre de données et nombre de paramètres n'est pas très élevé. Le critère NIC est aussi asymptotique. Il ne donne que des résultats quantitatifs grossiers (qu'il est superflu de raffiner en utilisant une statistique de test). La méthode de validation croisée pourra apporter un support supplémentaire à cette prise de décision.

2.3.5 Validation croisée

Il est sensé d'estimer l'erreur de prédiction à l'aide d'un ensemble de données n'ayant pas servi pour construire le prédicteur. Toutefois, utiliser un grand ensemble test peut apparaître comme une perte d'information lorsque le nombre des données est petit. Un petit ensemble test conduirait en revanche à des conclusions peu fiables. Voici une solution à ce problème.

Divisons l'ensemble des données $\mathcal{D}$ en deux moitiés (les (x_i, y_i) peuvent être choisis au hasard et rangés dans chaque moitié). On construit le prédicteur l'aide de la première moitié des données, puis on estime l'erreur de prédiction à l'aide de la seconde. On recommence cette opération en échangeant le rôle des deux moitiés. En moyennant, les deux résultats, on diminue la variance de l'erreur de prédiction estimée.

Un inconvénient de cette méthode est que l'on obtient ainsi une estimation de l'erreur de prédiction n'utilisant que la moitié des données. Pour faire mieux, on peut diviser l'ensemble $\mathcal{D}$ en m parties. Pour estimer l'erreur de prédiction $\mathcal{E}_{\mathrm{pr}}(\theta_0)$, on calculera

$$\frac{1}{m} \sum_{i=1}^{m} E_i(\hat{\theta}_{(i)})$$

où $E_i(\hat{\theta}_{(i)})$ est l'erreur de prédiction empirique associée à l'ensemble de validation i et $\hat{\theta}_{(i)}$ estimé à l'aide des données complémentaires. Cette procédure d'estimation de l'erreur de prédiction est appelée *validation croisée*. Elle est en générale coûteuse numériquement. Lorsque $m = n$ tend vers l'infini, cette erreur est proche de l'erreur de prédiction théorique. L'estimation doit être donc proche de celle obtenue par le critère NIC.

La procédure de choix de modèle s'obtient par comparaison des erreurs de prédiction estimées par validation croisée. On peut démontrer qu'asymptotiquement, le critère NIC et la validation croisée conduisent à sélectionner le même modèle.

Exercice 2.3.7 *(Difficile) Reprendre les arguments de la section précédente pour démontrer que l'erreur de prédiction obtenue par validation croisée (avec $m = n$) et le critère NIC sont équivalents lorsque n est grand.*

Dans la pratique, on choisit souvent $m \approx 10$. Cela peut être justifié théoriquement (Ripley, 1996, p. 70).

Commentaires. La validation croisée est en pratique appliquée en tirant au hasard avec remise dans l'échantillon pour fabriquer les sous-ensembles de validation. Supposons que l'on dispose de $n = 200$ données. La commande

```
> Indices <- sample( 10, 200, replace=T)
```

permet de créer un vecteur d'indices utile pour définir les sous-ensembles de validation. Ces sous-ensembles seront de taille aléatoire (en moyenne égale à 20).

2.3.6 TP3 : Validation croisée

L'objectif est l'implantation du critère de validation croisée pour les problèmes de régression. Informellement, l'algorithme s'écrit

```
La methode de validation croisee :

 1. creer un vecteur de longueur length(x) de nombres
    choisis au hasard entre 1 et 10
    #(les 10 blocs pour la validation croisee)

 2. Somme <- 0

 3. pour i de 1 a 10
          a. prendre le complementaire du bloc i
          b. ajuster le reseau a l'aide des donnees sur
             le complementaire du bloc i
          c. predire la valeur des donnees du bloc i
          d. calculer l'erreur E_i sur le bloc i
          e. Somme <- Somme + E_i
```

```
 4. retourner Somme
```

1. Comparer pour les jeux de données de régression non linéaire, les six modèles obtenus pour trois tailles différentes (3, 12, 30) et $\lambda = 0.01, 0.001$. Les résultats obtenus sont-ils cohérents avec ceux obtenus par NIC? L'estimation dépend du choix aléatoire des blocs. Comment diminuer la variance dûe à ce choix?
2. Reprendre l'algorithme de validation croisée avec des blocs de taille 1 (*leave-one-out*).
3. Comparer les estimations de l'erreur de prédiction obtenues par les critères NIC et validation croisée *leave-one-out* pour des jeux de données simulées. Faire varier la taille de l'échantillon et sa variance. On prend soin d'initialiser les algorithmes de calcul de manière identique.
4. Déterminer le choix de modèle pour le jeu `x1`, `y1`.

2.3.7 Exercice (Assez Difficile inf)

La commande `lm` permet d'ajuster des modèles linéaires de la forme

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \epsilon,$$

où l'on suppose que ϵ est un bruit centré, de variance σ^2. La syntaxe de la commande est la suivante

```
 modele <- lm( y ~ x1 + x2)
```

lorsque la réponse est y et que les variables explicatives sont `x1` et `x2`. L'ajustement des coefficients se fait par la méthode des moindres carrés. On récupère les valeurs estimées dans le vecteur `modele$coeff`. Pour fixer les idées, notons

```
> x <- runif(100)
> x0 <- rep(1,100)
> x1 <- x
> x2 <- x^2
```

et

```
> y <- g(x) + 0.2*rnorm(100)
```

Que représente la valeur `E + q`, en sortie des instructions suivantes :

```
>  mod <- lm( y~ x1 + x2 )
>  E <- sum( (y - fitted(mod))^2 )
>  X <- cbind(x0,x1,x2)
>  J <- t(X) %*% X
>  J <- solve(J)
>  long <- length(x) / 2
>  J <- long * J
```

```
> K <- var( 2* (y - fitted(mod)) *X)
> q <- sum( diag( K %*% J ) )
> E + q
```

Écrire une procédure de validation croisée pour ce modèle. Comparer l'estimation de l'erreur de prédiction avec la valeur de `E + q`.

2.4 Approximation universelle

Le résultat d'approximation universelle s'exprime ainsi.

Proposition 2.4.1 *Toute fonction continue f de $\mathbb{R}^p$ dans $\mathbb{R}^q$ peut être approchée uniformément sur tout compact par un réseau de neurones à sortie linéaire.*

L'approximation uniforme sur tout compact de f signifie que, pour tout sous-ensemble K fermé borné (compact) de $\mathbb{R}^p$, et $\epsilon > 0$, on peut trouver g telle que

$$\forall x \in K \,, \quad \|f(x) - g(x)\| < \epsilon \,.$$

La proposition repose sur une version (sophistiquée) du théorème de Stone-Weierstrass.

Proposition 2.4.2 *Toute fonction continue f de $\mathbb{R}^p$ dans $\mathbb{R}$ peut être approchée uniformément sur tout compact par un polynôme trigonométrique.*

On trouve dans (Ripley, 1996) la preuve complète du résultat. Nous nous limiterons, dans ce cours, à la démonstration du cas unidimensionnel. Cette situation correspond aux modèles de régression étudiés dans ce chapitre.

Proposition 2.4.3 *Toute fonction continue f de $[0,1]$ dans $\mathbb{R}$ peut être approchée uniformément par un réseau de neurones à sortie linéaire.*

Démonstration. La démonstration met l'accent sur l'aspect "non-constructif" du résultat d'approximation universelle.

Pas 1. Un monôme $\varphi(x) = x^r$ peut être approché uniformément par un réseau de neurones. Pour cela, on approche φ par une fonction étagée à $\epsilon/4$ près (une fonction étagée est une combinaison linéaire d'indicatrices d'intervalles). Or, une fonction indicatrice d'intervalle peut être approchée par une différence de fonctions logistiques correctement biaisées. On approche ainsi la combinaison linéaire par un réseau à $\epsilon/2$ près.

Pas 2. D'après le pas 1, un monôme φ peut être approché à $\epsilon/2$ près par un réseau de neurones à sortie linéaire. Il en va de même de tout polynôme (puisqu'un polynôme est combinaison linéaire de monômes). D'autre part, un polynôme peut être une approximation à $\epsilon/2$ près de f, d'après la version classique du théorème de Stone-Weierstrass. En recollant les erreurs, on arrive à une approximation de l'ordre de ϵ de f par un réseau de neurones.

■

Commentaires. La généralisation à q quelconque ne comporte pas difficulté (exercice). La généralisation à p quelconque est plus technique, et repose sur la forme générale des polynômes trigonométriques.

2.5 Convergence des réseaux de neurones

L'objet de cette section est la convergence (lorsque la taille n de l'échantillon grandit) d'un estimateur vers l'objet que l'on souhaite estimer. Dans cette section, nous justifions le fait que n'importe quelle fonction de régression

$$g(x) = E[Y|\, x]$$

assez régulière peut être estimée précisement à l'aide d'un réseau de neurones lorsque le nombre de données est suffisamment grand. La véritable démonstration est horriblement technique, et fait appel à des concepts délicats (comme la dimension de Vapnik-Cervonenkis). Nous nous contenterons de donner les grandes lignes. Notons que la vitesse de convergence d'un estimateur de type réseau de neurones est un sujet qui est encore très ouvert, et suscite de nombreuses recherches.

Nous observons n variables aléatoires

$$\mathcal{D}_n = \{(x_i, y_i)\}\ ,$$

et fabriquons un estimateur de la fonction de régression à l'aide d'un réseau de neurones. Supposons que ce réseau possède M cellules cachées. Cela revient à dire que l'estimateur appartient à une certaine classe de fonctions, que l'on note $\mathcal{F}_M$ (cette classe regroupe l'ensemble de tous les réseaux de neurones qui possèdent M cellules cachées). Notre estimateur est solution du problème de minimisation fonctionnelle

$$\hat{f}_M(.;n) = \arg\ \min_{f\in\mathcal{F}_M} \sum_{i=1}^{n} [y_i - f(x_i)]^2\ .$$

Commentaires. Résoudre ce problème de minimisation peut être très difficile en pratique. Toutefois, la discussion suivante concerne le véritable optimiseur, et non le résultat d'un algorithme particulier (comme la rétropropagation de l'erreur). Bien entendu, il serait plus réaliste de considérer la solution numérique (plutôt que la solution formelle). Mais il n'y a pas à notre connaissance d'outils analytiques assez poussés pour aborder ce problème.

Pour n suffisamment grand, nous souhaitons que $\hat{f}_M(x;n)$ soit proche de $E[Y|\ x]$. Notons f_M la meilleure approximation qu'il nous soit possible de trouver dans la famille $\mathcal{F}_M$

$$f_M = \arg \min_{f\in\mathcal{F}_M} E[(Y - f(X))^2]$$

c'est à dire la fonction de $\mathcal{F}_M$ qui conduira à la plus petite erreur de prédiction. Bien entendu, si $\mathcal{F}_M$ est trop petit, l'estimateur $\hat{f}_M(.;n)$ ne sera pas performant en général. Nous supposerons que $\mathcal{F}_M$ grossit lorsque $M \uparrow \infty$. Voici la trame principale de l'argument de convergence.

Pas 1. Vérifier que la classe $\mathcal{F}_M$ est suffisament riche. C'est à dire montrer qu'il existe une suite f_M telle que

$$E[(f_M(X) - E[Y|X])^2] \to 0 \quad \text{lorsque } M \to \infty\ .$$

Cela découle de l'approximation universelle pour les réseaux de neurones et une fonction de régression continue définie sur un intervalle $[a, b]$.

Pas 2. Etablir une loi des grands nombres uniforme (la principale difficulté de la démarche)

$$\lim_{n\to\infty} \sup_{f\in\mathcal{F}_M} \left| \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2 - E[(Y - f(X))^2] \right| = 0$$

en contrôlant la vitesse de convergence (c'est ici que certaines notions très techniques interviennent : citons la dimension de Vapnik de l'ensemble $\mathcal{F}_M$).

Pas 3. A l'aide du contrôle de la convergence, montrer que le résultat précédent est encore vrai lorsque $M(= M_n)$ et n tendent simultanément vers l'infini.

Pas 4. Assembler les pièces du puzzle. Tout d'abord, nous avons

$$\lim_{n\to\infty} E[(\hat{f}_{M_n}(X;n) - E[Y|X])^2] = \lim_{n\to\infty} E[(\hat{f}_{M_n}(X;n) - Y)^2] - E[(Y - E[Y|X])^2]\ .$$

D'après la loi des grands nombres uniforme (pas 3), cette limite est égale à

$$\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}_{M_n}(x_i;n))^2 - E[(Y - E[Y|X])^2]\ .$$

Par définition du problème de minimisation, pour tout f_{M_n}, nous avons

$$\lim \le \lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n}(y_i - f_{M_n}(x_i))^2 - E[(Y - E[Y|X])^2]$$

A nouveau, d'après la loi des grands nombres

$$= \lim_{n\to\infty} E[(f_{M_n}(X) - Y)^2] - E[(Y - E[Y|X])^2]$$

Ce dernier terme est égal à

$$\lim_{n\to\infty} E[(f_{M_n}(X) - E[Y|X])^2] = 0$$

grâce au pas 1. ■

Chapitre 3

Classification avec des réseaux de neurones

3.1 Théorie de la décision

Dans ce chapitre, nous nous intéressons à des problèmes de classification pour lesquels les classes sont prédéfinies. Les problèmes se présentent sous la forme suivante. Certains objets doivent classés selon leur type, ou classe, que nous supposons être étiqueté(e) par un entier compris entre 1 et K. Pour chaque objet, un certain nombre de mesures ont été effectuées. L'ensemble de ces mesures forme un vecteur X appelé *vecteur de caractéristiques*. Typiquement, le vecteur X appartient à une partie de $I\!R^p$ (s'il y a p caractéristiques distinctes).

L'objectif est de classer un objet. Cela signifie prendre une des $K+1$ décisions suivantes : $1, \ldots, K, D$ à partir de la valeur observée $X = x$. La décision D signifie "douter", et reporte la décision jusqu'à ce que de nouvelles informations puissent être prises en compte. Notons, qu'il est aussi possible de définir une décision R, pour "rejet", qui signifie que la variable n'appartient à aucune des classes. En pratique, cette décision détecte les données aberrantes. Mais elle est d'usage délicat et nous ne la considérerons pas dans ce cours.

En général, la véritable relation liant la classe C au vecteur X n'est pas forcément déterministe. Nous notons $p(x, k)$ la loi conjointe du couple (X, C), $p_k(x)$ la loi conditionnelle de X sachant $C = k$, et π_k la probabilité d'apparition de la classe k

$$\pi_k = \mathrm{P}(C = k) \ .$$

3.1.1 Règle de Bayes

Nous traitons le cas idéal où les π_k et les densités $p_k(x)$ sont connues. Cela permet de construire des règles de classement optimales. Comme lors du chapitre précédent, la tâche de classification consiste à prédire la variable C après avoir observé X. Considérons le classifieur ($\mathcal{X} \subset I\!R^p$)

$$\hat{c} : \mathcal{X} \to \{1, \ldots, K, D\} \ .$$

Un critère permettant de juger de la qualité de la classification est d'évaluer les probabilités conditionnelles de mal classer

$$\mathrm{pmc}(k) = \mathrm{P}(\hat{c}(X) \neq k\ ,\ \hat{c}(X) \neq D \,|\, C = k)$$

et les probabilités de doute

$$\mathrm{pd}(k) = \mathrm{P}(\hat{c}(X) = D \,|\, C = k)\ .$$

La formule des probabilités totales permet alors de calculer la probabilité de mal classer

$$\mathrm{pmc} = \mathrm{P}(\hat{c}(X) \neq C)$$

et la probabilité de doute

$$\mathrm{pd} = \mathrm{P}(\hat{c}(X) = D)\ .$$

A chaque mauvais classement est associé un coût élémentaire défini de la manière suivante.

Définition 3.1.1 *La fonction de coût L est définie de la manière suivante. Soit $k \in \{1, \ldots, K\}$ et $\ell \in \{1, \ldots, K, D\}$, alors*

$$L(k, \ell) = \begin{cases} 0 & si\ \ell = k; \\ 1 & si\ \ell \neq k\ \ et\ \ell \neq D; \\ d & si\ \ell = D; \end{cases}$$

où d est un nombre positif.

Notons que cette fonction de coût est utile pour comprendre les concepts importants, mais son utilisation pratique systématique peut-être désastreuse car elle suppose que toutes les mauvaises décisions sont équivalentes. Dans une application réelle, il sera possible de définir d'autres fonctions de coût, mesurant mieux le poids de chaque décision.

Définition 3.1.2 *La fonction de risque associée au classifieur $\hat{c}$ est le coût moyen sachant $C = k$*

$$R(\hat{c}, k) = E[L(k, \hat{c}(X)) \,|\, C = k]\ .$$

Le risque total est

$$R(\hat{c}) = E[R(\hat{c}, C)]\ .$$

Exercice 3.1.1 *Calculer $R(\hat{c}, k)$ et $R(\hat{c})$. On considère un échantillon*

$$\mathcal{D} = \{(X_i, C_i)\ ,\ i = 1, \ldots, n\}\ .$$

Quelle est la limite de

$$\frac{1}{n} \sum_{j=1}^{n} L(C_j, \hat{c}(X_j))\ .$$

Soit

$$p(k|\ x) = \mathrm{P}(C = k|\ X = x)$$

la probabilité (a posteriori) que la classe soit k sachant que l'on a observé $X = x$.

Proposition 3.1.1 Règle de Bayes. *La règle de classement qui minimise le risque total est*

$$c(x) = \begin{cases} k & si\ p(k|\ x) = \max_{\ell \leq K} p(\ell|\ x) > 1 - d; \\ D & si\ p(k|\ x) \leq 1 - d\ ,\ pour\ tout\ k \leq K; \end{cases}$$

Démonstration. Exercice ∎

Exercice 3.1.2 *Trouver la règle optimale de classement pour une fonction de coût quelconque.*

La règle optimale, définie par la proposition 3.1.1 s'appelle la *règle de Bayes*. En pratique, nous ne pouvons pas l'implanter car elle suppose que tout est connu. En revanche, la proposition 3.1.1 montre qu'il est utile d'estimer les probabilités conditionnelles $p(k|\ x)$.

Exercice 3.1.3 *On prend $K = 3$ et on suppose que la loi p_k est la loi de Poisson de paramètre λ_k, avec $\lambda_1 = 4$, $\lambda_2 = 8$ et $\lambda_3 = 15$. On suppose que $d = 1$. Déterminer la règle de Bayes (la loi π est uniforme).*

3.1.2 L'exemple gaussien

Supposons que la densité associée à la classe k est la densité de la loi normale de moyenne μ_k et de matrice de covariance Σ, indépendante de k

$$\forall x \in I\!R^p\ , \quad p_k(x) = \frac{1}{(2\pi)^{p/2}\sqrt{det(\Sigma)}} e^{-q_k(x)/2}$$

où $q_k(x) = (x - \mu_k)^T\, \Sigma^{-1}(x - \mu_k)$.

Définition 3.1.3 *On appelle distance de Mahanalobis la grandeur*

$$d(x, \mu) = [(x - \mu)^T \Sigma^{-1}(x - \mu)]^{1/2}$$

Commentaires. Cette notion, datant de 1936, mesure la distance séparant un point x du centre d'une classe, lorsqu'elle est appliquée à $\mu = \mu_k$.

Exercice 3.1.4 *Montrer que la règle de Bayes consiste à trouver k tel que la grandeur*

$$-2\mu_k^T \Sigma^{-1} x + \mu_k^T\, \Sigma^{-1} \mu_k - 2 \log \pi_k$$

soit minimale. Cette règle s'appelle analyse discriminante linéaire.

Exercice 3.1.5 *Supposons que le nombre de classes se réduise à*

$$K = 2\ .$$

1) *Montrer que l'on choisit la classe 1 lorsque*

$$\log \pi_1 - \frac{1}{2} d(x, \mu_1)^2 > \log \pi_2 - \frac{1}{2} d(x, \mu_2)^2\ ,$$

ou de manière équivalente, lorsque

$$A = (\mu_1 - \mu_2)^T \Sigma^{-1} (x - \bar{\mu}) > \log(\pi_2/\pi_1)$$

avec $\bar{\mu} = (\mu_1 + \mu_2)/2$.

2) *On suppose que* X *suit la loi de densité* $p_1(x)$. *Montrer que la loi de* A *est la loi* $\mathcal{N}(d^2/2, d^2)$ *où*

$$d = d(\mu_1, \mu_2)\ .$$

3) *En déduire le risque associé à la règle de Bayes.*

3.2 Réseaux de neurones

3.2.1 Modèles classiques

Comme nous l'avons vu dans la section précédente, nous avons besoin d'estimer la probabilité pour que le type soit k sachant que l'on observe x. Le modèle proposé par un réseau de neurones est

$$p(k;\, x, \theta) = \phi_o \left(\alpha^k + \sum_h w_h^k\, \phi_h \Big(\alpha_h + \sum_j w_h^j\, x^j \Big) \right) \tag{3.1}$$

pour tout $k \in \{1, \ldots, K\}$. Par définition, la couche d'entrée comporte p cellules (ceci correspond à la dimension du vecteur x), et la couche de sortie comporte K cellules. Comme on le verra plus tard, il est possible d'utiliser $K-1$ cellules en sortie (et de tenir compte du fait que les probabilités se somment à 1) en utilisant l'option *softmax* de la commande *nnet*. La fonction ϕ_o doit varier entre 0 et 1. Le logiciel propose d'utiliser la fonction logistique

$$\ell(z) = \frac{1}{1 + \exp(-z)}\ .$$

Toutefois, on pourrait utiliser n'importe quelle fonction de répartition. Pour ajuster le paramètre θ, nous disposons d'un ensemble de données

$$\mathcal{D} = \{(x_i, c_i)\}$$

composé de caractéristiques et de classes associées. Les probabilités conditionnelles $p(k|\, x)$ sont estimées en transformant chaque étiquette de classe k en le vecteur unitaire e_k, ou de manière équivalente en l'indicatrice de la classe k

$$e_k(\ell) = \mathbb{1}_k(\ell)\ , \quad \text{pour } \ell \in \{1, \ldots, K\}.$$

L'ajustement peut se faire à l'aide du critère des moindres carrés. Toutefois, pour implanter le critère de choix de modèle, il est préferable d'utiliser l'entropie (section 3.3).

3.2.2 Régression logistique

Le modèle de régression logistique est une alternative évidente (et simple) à l'utilisation de réseaux de neurones. Nous le présentons dans le cas de deux classes

$$K = 2 \; .$$

Cette démarche consiste à écrire un modèle *log-linéaire*

$$\log p(2|\ x) = \log p(1|\ x) + \alpha + \beta^T x \; . \tag{3.2}$$

Puisque la somme des probabilités est égale à 1, nous avons

$$p(2|\ x) = \ell(\alpha + \beta^T x)$$

où ℓ est la fonction logistique. Ce modèle correspond donc à un réseau de neurones sans couche cachée, mais où les entrées sont directement liées à la cellule de sortie (option `skip = T`). Sous S-Plus, l'estimation des paramètres s'obtient grâce à la commande `nnet` avec l'option softmax, le paramètre size = 0, et le booléen skip fixé à la valeur T. Mais il est aussi possible d'utiliser la commande `glm` qui utilise un optimiseur différent de la rétropropagation. L'estimation des paramètres se fait toujours par la méthode de maximum de vraisemblance. La syntaxe est la suivante

```
> log.reg <- glm( c ~ x, family = binomial, maxit = 50)
```

La syntaxe `c   x` est commune aux modèles de régression. Elle signifie que l'on souhaite "régresser" la variable binaire c par la variable x. L'option `family = binomial` signifie que la loi de c est une loi binomiale $\mathcal{B}(1, p)$ (loi de Bernoulli). Le détail de la commande `glm` dépasse le cadre de ce cours. Notons tout de même que les fonctions `fitted.values`, et `predict` s'appliquent aux objets définis par cette commande.

Pour terminer ce paragraphe, il est intéressant de noter une analogie d'expression avec l'analyse discriminante, étudiée dans le paragraphe 3.1.2, exercice 3.1.5. Reprenons les notations de ce paragraphe. Nous obtenons

$$2\frac{\log p(2|\ x)}{\log p(1|\ x)} = 2(\mu_2 - \mu_1)^T \Sigma^{-1} x - (\mu_2 + \mu_1)^T \Sigma^{-1} (\mu_2 - \mu_1) + 2 \log \frac{\pi_2}{\pi_1} \; .$$

Il s'agit d'une équation linéaire, analogue à l'equation 3.2. Ainsi il peut paraître équivalent d'utiliser la régression logistique ou l'analyse discriminante linéaire pour implanter la règle de Bayes. Toutefois il y a une différence fondamentale. La régression logistique estime "directement" la probabilité conditionnelle $p(2|\ x)$, alors que, pour l'analyse discriminante, on estime μ_1, μ_2, Σ pour fabriquer une estimation "indirecte" de $p(2|\ x)$. En pratique, les deux méthodes peuvent conduire à des résultats parfois très différents.

Exercice 3.2.1 *Tracer les frontières de décision pour les données artificielles créées lors du second TP en utilisant la régression logistique. Faire de même pour l'exemple traitant des crabes.*

3.3 Critères de sélection

Comme pour le cas de la régression non-linéaire, nous étudierons deux méthodes pour la comparaison de modèles : le critère NIC et la validation croisée. Les résultats se présentent de manière analogue au chapitre précédent. En voici les différents étapes sous forme d'exercice.

3.3.1 Critère NIC

Exercice 3.3.1 *On considère un problème de classification comportant deux classes et un échantillon de données de taille n. Déterminer la log-vraisemblance du paramètre θ conditionnelle aux x_i. Généraliser à K classes.*

Exercice 3.3.2 *Calculer les dérivées partielles de l'erreur d'entropie par rapport au paramètre θ.*

Exercice 3.3.3 *Donner une définition du critère NIC dans le cas où l'on utilise l'erreur d'entropie. Et dans le cas des moindres-carrés ?*

Exercice 3.3.4 *Ecrire une procédure en S-Plus pour calculer le critère NIC en classification.*

3.3.2 Validation croisée

La procédure de validation croisée permet de comparer différents modèles (de manière plus fiable que le critère NIC en général). Il n'y a pas de modification majeure à apporter à la méthode vue au chapitre précédent. La seule variante concerne le critère de prédiction. Il est possible d'utiliser l'erreur d'entropie, mais il est plus simple d'estimer la probabilité d'erreur "pmc" de la méthode choisie. Cela revient à compter les mal classés dans les sous-ensembles de validation.

Notons que la procédure de validation croisée permet aussi de fixer les paramètres d'un réseau de neurones qui ne sont pas estimés lors de la rétropropagation. Par exemple, cela permet de fixer le paramètre λ ou le paramètre "maxit".

3.3.3 TP4 : Choix de modèle en classification

L'objectif de ce TP est l'implantation du critère de validation croisée pour les problèmes de classification. Ecrire une procédure de validation croisée en S-Plus pour un problème à K classes et un échantillon

$$\mathcal{D} = \{(x_i, c_i)\}$$

de taille n ($= 200$), que l'on partitionnera en 10 sous-ensembles (de tailles voisines).

Sélectionner un modèle "convenable" pour les trois jeux de données étudiés au TP2 ($\lambda = 0.1, 0.01$). Pour l'exemple des crabes, quel modèle donne les meilleurs résultats : le modèle logistique, ou un réseau de neurones? Utiliser toutes les caractéristiques pour ce jeu de données.

Tracer les frontières de décision obtenues par la règle de Bayes pour les modèles qui vous semblent les meilleurs dans les trois exemples de classification. Moyenner la prédiction en répétant l'algorithme *nnet* avec des initialisations différentes. Tracer les nouvelles frontières de décision.

Références bibliographiques

J. Aitchison, et I.R. Dunsmore. **Statistical Prediction Analysis**, Cambridge University Press, 1975.

G.E. Hinton. *Learning distributed representation of concepts.* In **Proceedings of the 8th Annual Conference of the Cognitive Science Society**, 1-12, Hillsdale: Erlbaum, 1986.

M. Hornik, M. Stinchcombe, et H. White. *Multilayer feedforward networks are universal approximators*, Neural Networks, **2**, 359-366, 1989.

S. Geman, E. Bienenstock, et R. Doursat. *Neural networks and the bias/variance dilemma*, Neural Computation, **4**, 1-58, 1992.

U. Grenander. *On empirical spectral analysis of stochastic processes*, Ark. Matemat., **1**, 503-531, 1951.

B.D. Ripley. **Pattern Recognition and Neural Networks**, Cambridge University Press, 1996.

B.D. Ripley. *Neural networks and related methods for classification (with discussion)*, Journal of the Royal Statistical Society B, **56**, 409-456, 1994.

D.E Rumelhart, G.E. Hinton, et R.J. Williams. *Learning representation by backpropagating errors*, Nature (London), **323**, 533-536, 1986.

W.N. Venables, et B.D. Ripley. **Modern Applied Statistics with S-Plus**. New-York: Springer, 1994.

W.N. Venables, et D. Smith. *Notes on S-Plus: A programming environment for data analysis and graphics.* Document électronique, 1992.

G. Wahba. *Generalization and regularization in nonlinear learning systems.* In **The Handbook of Brain Theory and Neural Networks**, M. Arbib ed., 426-430, Cambridge MA: The MIT press, 1995.

G. Wahba, et S. Wold. *A completely automatic french curve*, Communications in Statistics, **4**, 1-17, 1975.

P. Werbos. **Beyond Regression: New Tools for Prediction and Analysis in the Behavioural Sciences**. PhD Thesis, Harvard University, 1974.

H. White. *Learning in artificial neural networks: a statistical persepective*, Neural Computation, **1**, 425-464, 1989.

H. White. *Some asymptotic results for learning in single hidden-layer feedforward networks*, Journal of the American Statistical Association, **84**, 1003-1013, 1989.