Validation croisée pour le choix de paramètre de méthodes de régularisation

julien.chiquet@genopole.cnrs.fr

Module MPR – option modélisation, 23 décembre 2009

Table des matières

1 Motivations

Soit le modèle linéaire défini par

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon},$$

où $\mathbf{y}$ est un vecteur de réponse contenant n observations et $\mathbf{X}$ la matrice des prédicteurs de taille $n \times p$. On suppose que l'intercept β_0 est estimé par $\bar{\mathbf{y}} = n^{-1}\sum_{i=1}^{n} y_i$, que le vecteur $\mathbf{y}$ est centré et que la matrice $\mathbf{X}$ est centrée-réduite. Ainsi l'estimation ne porte plus que sur le vecteur $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)^\intercal$ de $\mathbb{R}^p$.

Les méthodes de régularisation telles la régression *ridge*, le Lasso, l'*Elastic-net*, etc. peuvent s'écrire

$$\widehat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}\in\mathbb{R}^p} \left\{ \frac{1}{2}\mathrm{RSS}(\boldsymbol{\beta}) + \lambda \cdot \mathrm{pen}(\boldsymbol{\beta}) \right\}, \tag{1}$$

où $\mathrm{pen}(\boldsymbol{\beta})$ est une fonction pénalisant les coefficients de $\boldsymbol{\beta}$ et où

$$\mathrm{RSS}(\boldsymbol{\beta}) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 .$$

Cette note a pour objectif de proposer une méthode simple de choix du paramètre λ, se fondant sur la minimisation de l'erreur de prédiction estimée par validation croisée.

2 Une méthode de choix de λ

2.1 Erreur de prédiction, erreur de test

Définition (Erreur de prédiction)**.** *On appelle erreur de prédiction du modèle de régression*

$$y = f(X) + \varepsilon$$

la grandeur définie par

$$\mathrm{EPE}(f) = \mathbb{E}[(y - f(X))^2]. \tag{2}$$

Il est naturel de vouloir évaluer l'erreur de prédiction d'un modèle statistique. Une première (mauvaise) idée, qui a tendance à sous évaluer l'erreur de prédiction, est de comparer le modèle $\hat{f}$ estimé sur un jeu de donnée avec les valeurs des réponses $\mathbf{y}$. Dans le cas du modèle linéaire, le modèle s'écrit $\hat{f}(\mathbf{X}) = \mathbf{X}\hat{\boldsymbol{\beta}}$. On estime l'erreur de prédiction en moyennant les distances entre les valeurs réelles y_i et les valeurs $\mathbf{X}_i\hat{\boldsymbol{\beta}}$ estimées par le modèle :

$$\mathrm{err}(f) = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{y}_i - \mathbf{X}_i\hat{\boldsymbol{\beta}})^2, \tag{3}$$

où $\mathbf{X}_i$ est la i^{e} ligne de la matrice des prédicteurs, correspondant au i^{e}individu.

Le problème de ce calcul est que l'on utilise les mêmes données pour estimer les paramètres du modèle et pour évaluer l'erreur commise : il est normal que $\mathbf{X}_i\hat{\boldsymbol{\beta}}$ prédise au mieux la valeur de y_i, puisqu'on a utilisé $y_i, \mathbf{X}_i$ pour estimer $\boldsymbol{\beta}$. Le modèle ainsi construit risque fortement de se généraliser assez pauvrement à un autre jeu de données correspondant au même phénomène : il sera extrêmement bien adapté aux données $\mathbf{y}, \mathbf{X}$ mais se comportera mal sur un jeu $\mathbf{y}', \mathbf{X}'$ obtenus pour des individus différents. On parle de *surapprentissage*.

Pour éviter ce problème, la solution idéale dans le cas d'un jeu de données où le nombre d'observations n est suffisamment grand consiste à découper artificiellement les données en deux ensembles tirés aléatoirement : un ensemble d'apprentissage, composé des deux tiers des données sur lequel on va estimer le modèle $\hat{f}$, et un ensemble de test composé du tiers restant sur lesquels on va tester le bon comportement du modèle.

Notons $\hat{\boldsymbol{\beta}}^{\mathrm{appr}}$ le modèle estimé sur la partie des données étiquetée comme *ensemble d'apprentissage* et $\mathbf{y}^{\mathrm{test}}$. L'*erreur de test*, définie par

$$\mathrm{err}_{\mathrm{test}}(f) = \frac{1}{n_{\mathrm{test}}}\sum_{i=1}^{n_{\mathrm{test}}}(\mathbf{y}_i^{\mathrm{test}} - \mathbf{X}_i^{\mathrm{test}}\hat{\boldsymbol{\beta}}^{\mathrm{appr}})^2, \tag{4}$$

estime correctement l'erreur de prédiction lorsque n est grand. C'est l'erreur que nous avons systématiquement calculer dans les notes précédentes.

2.2 Validation croisée

En pratique, les jeux de données sont rarement suffisamment grands pour que l'erreur de test ainsi calculée estime correctement l'erreur de prédiction du modèle. La validation croisée est une alternative très populaire pour gérer la parcimonie des données. Il s'agit de découper le jeu de données en K groupes tirés aléatoirement qui font successivement servir d'ensemble de test. On peut alors calculer une erreur de test pour chacun des groupes et en faire la moyenne, ce qui constitue l'estimateur de l'erreur de test par validation croisée. Notons $\mathrm{CV}(f)$ cette erreur pour un modèle f donné.

On introduit également une fonction $\kappa : \{1, \ldots, n\} \to \{1, \ldots, K\}$ qui indique dans quel groupe de validation se trouve l'observation i. Ainsi, $\kappa(i) = k$ si l'observation i se trouve dans le groupe k de validation. Si on note $\hat{f}^{-k}$ le modèle estimé sur l'ensemble des données privé du k^{e} groupe, alors l'erreur de validation croisée s'écrit

$$\mathrm{CV}(f) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{f}^{-\kappa(i)}(\mathbf{X}))^2. \tag{5}$$

Ainsi, y_i, la réponse observée, est à comparer avec $\hat{f}^{-\kappa(i)}(\mathbf{X}) = \hat{y}_i^{-\kappa(i)}$, la valeur prédite du modèle estimé sur les données privés de l'ensemble de test où se trouve l'observation i. Lorsque $K = n$, on retrouve le cas extrême correspondant au *leave-one-out* (ou laissé pour compte)[1].

3 Code `R` commenté

Les données prostate contiennent une matrice `x`, un vecteur `y`. On charge les données et les fonctions qui sont stockées dans les fichiers `functions.R` et `functions_cv.R`, puis on centre et on réduit la matrice $\mathbf{X}$.

```
> rm(list = ls())
> library(glmnet)
> source("functions.R")
> source("functions_cv.R")
> load("prostate.rda")
> x <- scale(as.matrix(x))
```

3.1 Validation croisée

Dans la suite, nous allons calculer l'erreur du *leave-one-out* pour 50 valeurs de λ. On note `ng` le nombre de groupes (que vous pourrez changer à volonté).

```
> n.lambda <- 50
> ng <- nrow(x)
```

[1] Pour plus de détail voir le chapitre 7, en particulier à la section 7.10 du livre *The Elements of Statistical Learning*

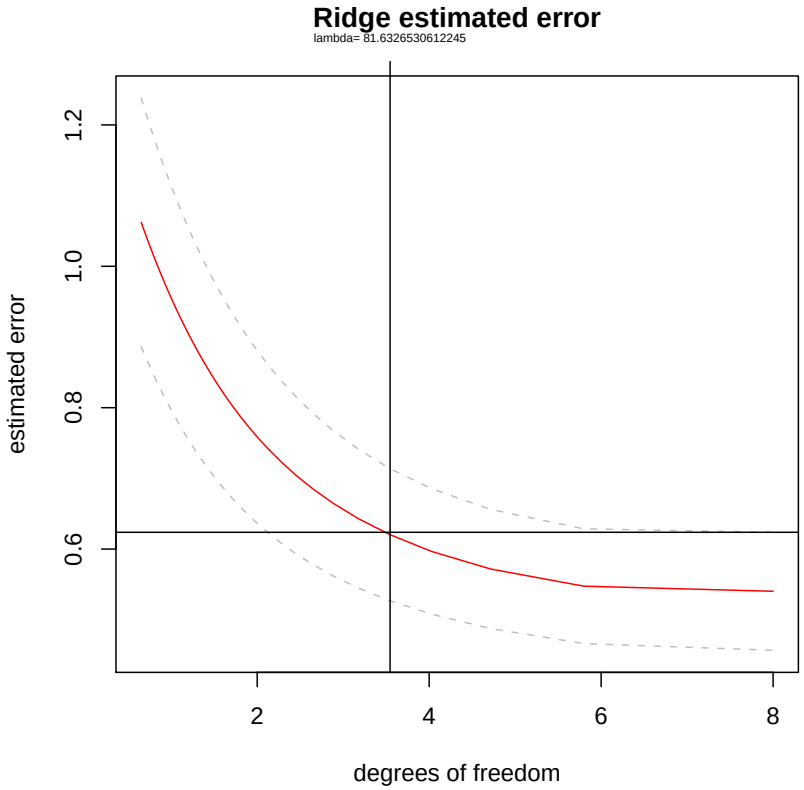
3.1.1 Régression ridge

On génère une séquence de λ entre 0 et 1000 puis on calcule l'erreur à l'aide de la fonction `cross.validation` détaillée plus loin

Une bonne heuristique pour le choix de λ consiste à faire un compromis entre l'erreur commise et la complexité du modèle. Nous choisissons la valeur de λ de la sorte : on repère la plus petite erreur de test, et le niveau supérieur de l'intervalle de confiance correspondant. On regarde alors quelles erreurs, pour les différents λ, sont plus petite que cette borne supérieur. On choisit le λ correspondant au modèle le moins complexe, comme représenté à la figure suivante [2] :

```
> matplot(out.r$abscisses, out.r$error, type = "l", col = c("gray",
+     "red", "gray"), lty = c(2, 1, 2), xlab = "degrees of freedom",
+     ylab = "estimated error", main = "Ridge estimated error")
> ind.min.err <- which.min(out.r$error[, 2])
> max.min.err <- out.r$error[ind.min.err, 3]
> i <- which(out.r$error[, 2] <= max.min.err)
> i.star <- min(i)
> abline(h = max.min.err)
> abline(v = out.r$abscisses[i.star])
> lambda.star <- lambda.r[i.star]
> axis(3, at = out.r$abscisses[i.star], labels = paste("lambda=",
+     lambda.star), cex.axis = 0.5)
```

[2] le code pour le tracé des graphes est laissé pour la compréhension

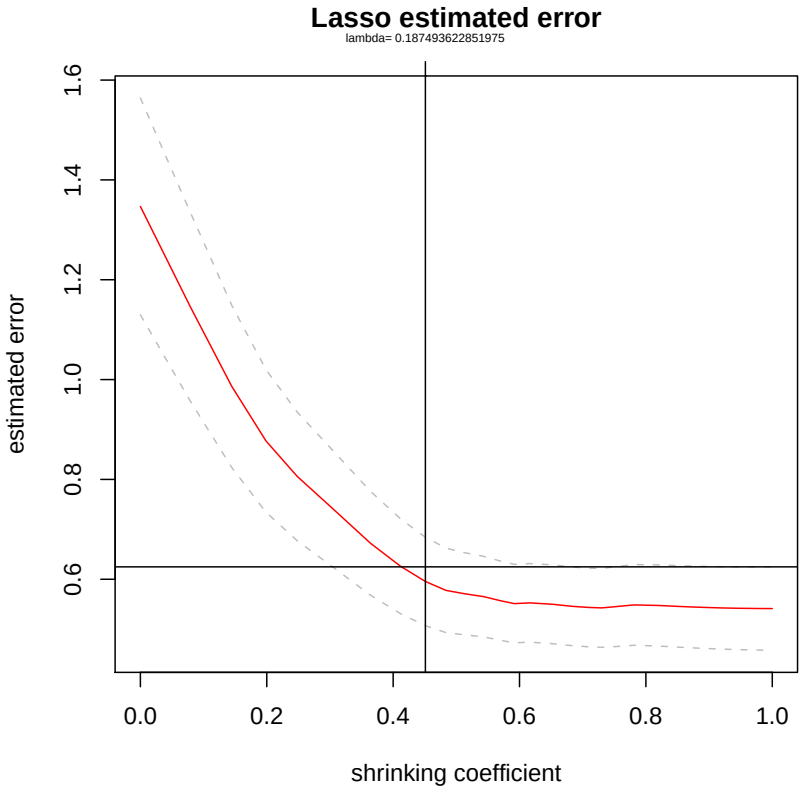


3.1.2 Lasso

Pour le LASSO, on utilise la séquence de λ générés par le package `glmnet`, puis on calcule l'erreur de test par validation croisée pour les valeurs correspondante.

```
> out <- glmnet(x, y, nlambda = n.lambda)
> lambda.l <- out$lambda
> out.l <- cross.validation(x, y, lambda.l, method = "lasso", ngroup = ng)
```

On obtient graphiquement



4 Code des fonctions

4.1 La fonction `cross.validation`

Cette fonction prend en argument les données `x` et `y`, un vecteur `lambda` de valeurs *décroissantes* du paramètre λ, une chaîne de caractère `method` pouvant prendre les valeurs `c("ridge","lasso","elnet")`[3] ainsi qu'un scalaire `ngroup` correspondant au nombre de groupes pour la validation croisée (par défaut n, c'est-à-dire le *leave-one-out*).

La fonction renvoie une liste contenant deux éléments : une matrice `cv.error` contenant 3 colonnes et autant de lignes que d'éléments dans `lambda` : la deuxième colonne correspond à l'estimateur de l'erreur de validation croisée et les deux autres les valeurs permettant de construire l'intervalle de confiance autour de cette valeur. On revoie également un vecteur d'abscisses pour tracer cette erreur, correspondant au niveau de complexité de la méthode. Pour la régression ridge, ce sont les degrés de liberté ; pour les méthodes du Lasso et l'*Elastic-Net*, il s'agit du niveau de rétrécissement du vecteur β renormalisé, soit, pour une valeur de λ,

$$\frac{\sum_{i=1}^{p} |\hat{\boldsymbol{\beta}}_i|}{\sum_{i=1}^{p} |\hat{\boldsymbol{\beta}}_i^{\max}|},$$

[3] À vous de l'enrichir si vous avez besoin d'autres méthodes...

valeur variant entre 0 (modèle le moins complexe, correspondant à la plus forte valeur de λ) et 1 (modèle le plus complexe, correspondant à la plus petite valeur de λ).

```
> cross.validation <- function(x, y, lambda, method = "ridge",
+     ngroup = nrow(x)) {
+     require(bootstrap)
+     if (method == "ridge") {
+         theta.fit <- theta.fit.ridge
+         theta.predict <- theta.predict.ridge
+         abscisses <- theta.fit(x, y, lambda)$df
+     }
+     if (method == "lasso") {
+         theta.fit <- theta.fit.lasso
+         theta.predict <- theta.predict.lasso
+         beta <- theta.fit(x, y, lambda)$beta
+         abscisses <- colSums(abs(beta))/max(colSums(abs(beta)))
+     }
+     cv.error <- c()
+     cat("\nLambda =")
+     for (l in lambda) {
+         cat("", l)
+         r <- crossval(x, y, theta.fit, theta.predict, lambda = l,
+             ngroup = ngroup)
+         err <- (y - r$cv.fit)^2
+         sd.err <- sd(err)/sqrt(length(err))
+         cv.error <- rbind(cv.error, c(mean(err) - sd.err, mean(err),
+             mean(err) + sd.err))
+     }
+     return(list(error = cv.error, abscisses = abscisses))
+ }
```

4.2 Les fonctions `theta.fit`, `theta.predict` des différentes méthodes

Les fonctions de fitting (calcul de l'estimateur $\hat{\boldsymbol{\beta}}$) et de prédiction (valeurs prédite de la réponse par le modèle) pour la ridge et le Lasso : attention à l'intercept ! (les données n'ont pas été centrées en y, donc ne pas l'oublier pour la fonction de prédiction).

```
> theta.fit.ridge <- function(x, y, lambda) {
+     ridge.regression(x, y, lambda)
+ }
> theta.predict.ridge <- function(fit, x) {
+     fit$beta0 + x %*% t(fit$beta)
+ }
> theta.fit.lasso <- function(x, y, lambda) {
+     glmnet(x, y, lambda = lambda)
+ }
```

```
> theta.predict.lasso <- function(fit, x) {
+     fit$a0 + x %*% as.numeric(fit$beta)
+ }
```