Étude de quelques méthodes régularisées pour la régression linéaire – Éléments de correction

julien.chiquet@gmail.com

séminaire professionel ENSAI - 2016

1 Mise au point d'un protocole de simulation

1.1 V1: modèle linéaire simple

Écrire une fonction `rlm(n,p,p0,sigma)` qui renvoie une liste contenant un vecteur `y`, une matrice `x`, un vecteur `beta` avec p_0 entrées non nulles de magnitude choisie selon une loi uniforme entre 1 et 2 et de signe positif ou négatif. Les prédicteurs sont tirés selon des distribution gaussiennes univariés indépendantes de variance unitaire.

```
rlm <- function(n, p, p0, sigma) {
  ## vecteur des vrais coefficients
  beta <- numeric(p)
  A <- sample(1:p, p0)
  beta[A] <- runif(p0, 1, 2)
  ## bruit blanc gaussien
  noise <- rnorm(n,0,1)
  ## prédicteurs sans structure particulière
  X <- matrix(rnorm(n*p), n, p)
  ## vecteur d'observation des réponses
  y <- X %*% beta + sigma * noise

  list(y=y, X=X, beta=beta)
}
```

1.2 V2: intégration du coefficient de détermination

Déterminer dans le cadre du modèle (**??**) quelle valeur de σ^2 choisir pour obtenir des données ayant un coefficient de détermination donné.

Soit $\mathbb{E}[XX^\top] = \Sigma$. Alors,

$$r^2 = \mathbb{E}(R^2) = 1 - \frac{SCR}{SCT} = 1 - \frac{\mathbb{E}\|\boldsymbol{\varepsilon}\|^2}{\mathbb{E}\|\boldsymbol{\varepsilon}\|^2 + \mathbb{E}\|\mathbf{X}\boldsymbol{\beta}\|^2} = 1 - \frac{n\sigma^2}{n\sigma^2 + \boldsymbol{\beta} n \Sigma \boldsymbol{\beta}} \quad (1)$$

En isolant σ^2

$$\sigma^2 = \frac{(1-r^2)}{r^2}\boldsymbol{\beta}\Sigma\boldsymbol{\beta} \quad (2)$$

En particulier, si $\Sigma = I_p$, on a

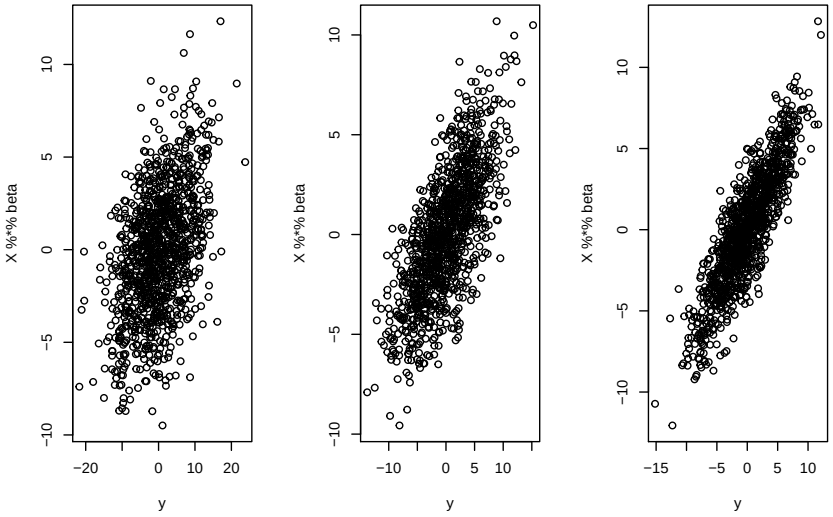
$$\sigma^2 = \frac{(1-r^2)}{r^2}\|\boldsymbol{\beta}\|^2 \quad (3)$$

Amender votre fonction en conséquence: elle prend désormais en argument `rlm(n,p,p0,r2)` et renvoie une liste `list(y,x,beta,sigma)`.

```
rlm <- function(n.train, p, p0, r2, n.test=10*n.train) {
  ## génération ensemble test/apprentissage
  n <- n.train + n.test
  train <- sample(1:n, n.train)
  test  <- setdiff(1:n, train)

  ## vecteur des vrais coefficients
  beta <- numeric(p)
  A <- sample(1:p, p0)
  beta[A] <- runif(p0, 1, 2)
  ## bruit blanc gaussien
  noise <- rnorm(n,0,1)
  ## prédicteurs sans structure particulière
  X <- matrix(rnorm(n*p), n, p)
  ## vecteur d'observation des réponses
  sigma <- sqrt( (1-r2)/r2 * t(beta) %*% cov(X) %*% beta)
  y <- X %*% beta + sigma * noise

  list(y=y, X=X, beta=beta, sigma=sigma, train=train, test=test)
}
```

Test de la fonction

```
data1 <- rlm(100,p=20,p0=5,r2=.25)
data2 <- rlm(100,p=20,p0=5,r2=.5)
data3 <- rlm(100,p=20,p0=5,r2=.75)
par(mfrow=c(1,3))
with(data1, plot(y, X%*%beta))
with(data2, plot(y, X%*%beta))
with(data3, plot(y, X%*%beta))
```



1.3 V3: structure de dépendance des prédicteurs

Écrire deux fonctions `rlm.long(n, p, p0, r2, rho)` et `rlm.bloc(n, p, K0, r2, rho, K)` adaptées à ces deux scénarios.

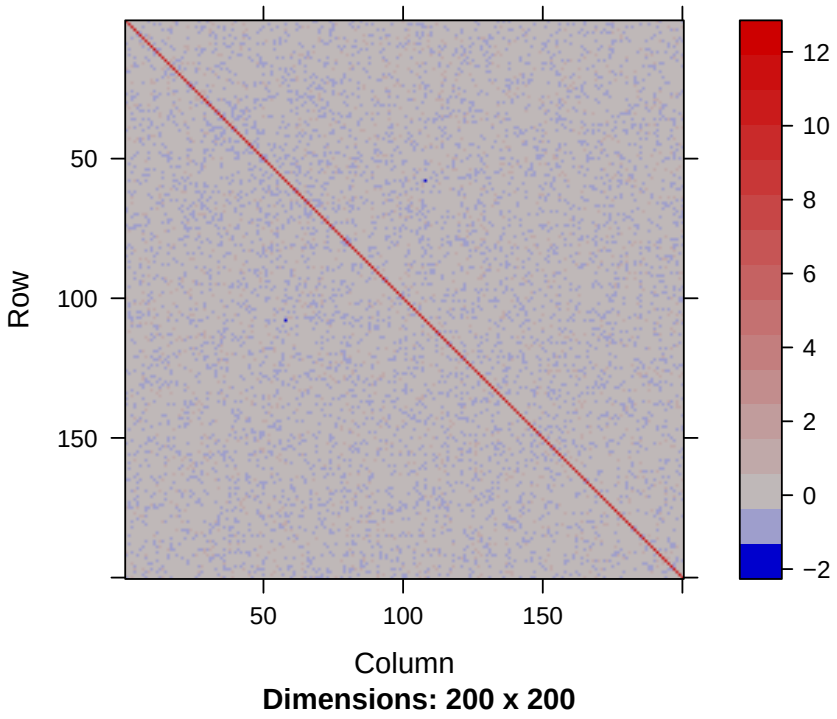
```
rlm.long <- function(n.train, p, p0, r2, rho, n.test=10*n.train) {
  ## génération ensemble test/apprentissage
  n <- n.train + n.test
  train <- sample(1:n, n.train)
  test  <- setdiff(1:n, train)

  ## vecteur des vrais coefficients
  beta <- numeric(p)
  A <- sample(1:p, p0)
  beta[A] <- runif(p0, 1, 2)
  ## bruit blanc gaussien
  noise <- rnorm(n,0,1)
  ## prédicteurs avec structure longitudinale
  Sigma <- toeplitz(rho^(0:(p-1)))
  X <- matrix(rnorm(n*p), n, p) %*% chol(Sigma)
  ## vecteur d'observation des réponses
  sigma <- sqrt( (1-r2)/r2 * t(beta) %*% cov(X) %*% beta)
  y <- X %*% beta + sigma * noise

  list(y=y, X=X, beta=beta, sigma=sigma, train=train, test=test)
}
```

```
library(Matrix)
rlm.bloc <- function(n.train, p, K0, r2, rho, K, n.test=10*n.train) {
  ## génération ensemble test/apprentissage
  n <- n.train + n.test
  train <- sample(1:n, n.train)
```

```
  test  <- setdiff(1:n, train)
  ## prédicteurs avec structure par bloc
  nk <- p/K
  Sigma <- bdiag(rep(list(matrix(rho,nk,nk)), K))
  diag(Sigma) <- 1
  X <- matrix(rnorm(n*p), n, p) %*% chol(Sigma)

  ## vecteur des vrais coefficients
  support.grp <- sample(rep(c(1,0), c(K0, K-K0)))
  support.beta <- rep(support.grp, each=nk)
  beta <- runif(p, 1, 2) * support.beta
  ## bruit blanc gaussien
  noise <- rnorm(n,0,1)
  ## vecteur d'observation des réponses
  sigma <- sqrt( (1-r2)/r2 * t(beta) %*% cov(as.matrix(X)) %*% beta)
  y <- X %*% beta + sigma * noise

  list(y=y, X=X, beta=beta, sigma=sigma, train=train, test=test)
}
```

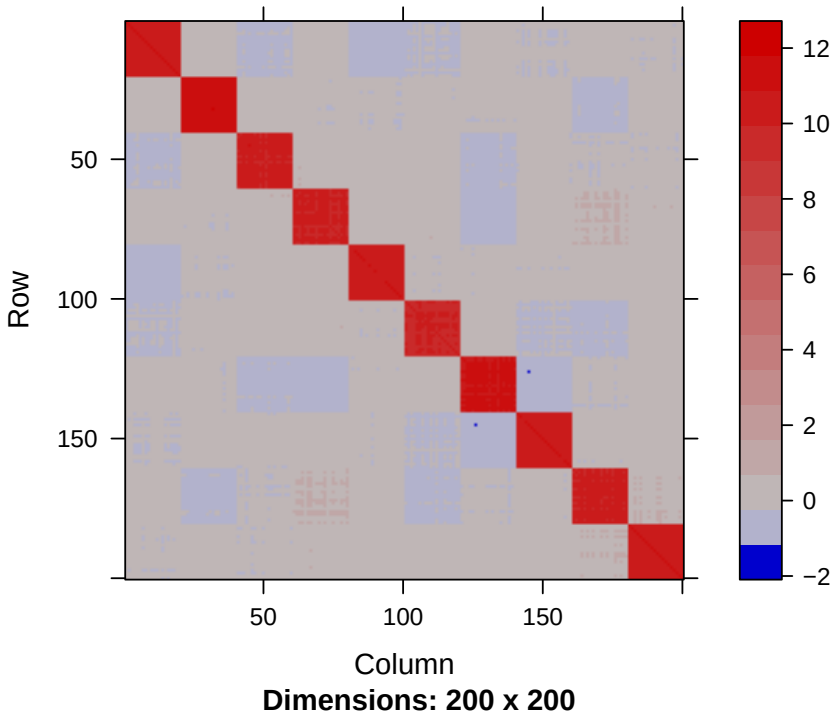
On dispose finalement de 3 fonctions `rlm.indep`, `rlm.long` et `rlm.bloc` renvoyant une liste contenant les variables `y,x,beta,Sigma,sigma,train,test` qui serviront à la génération de données pour les simulations.

```
n <- 100
p <- 200
rho <- .95
p0 <- 10
K0 <- 5
K <- 10
r2 <- 0.75
data.indep <- rlm(n, p, p0, r2)
data.long  <- rlm.long(n, p, p0, r2, rho)
data.bloc  <- rlm.bloc(n, p, K0, r2, rho, K)
image(Matrix(crossprod(data.indep$X)/n), useRaster=TRUE)
```



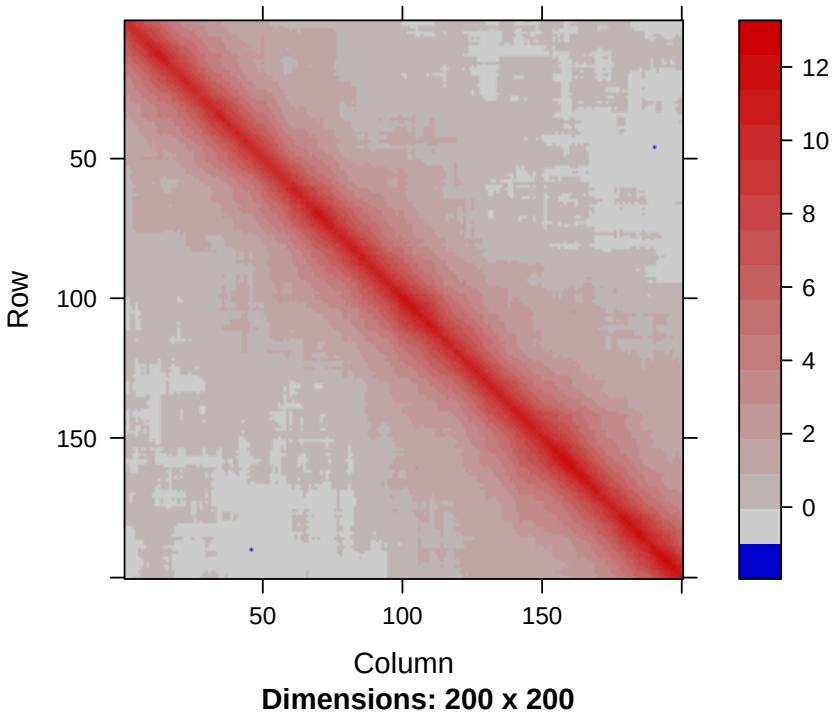
Dimensions: 200 x 200

```
image(Matrix(crossprod(data.bloc$X)/n), useRaster=TRUE)
```



Dimensions: 200 x 200

```
image(Matrix(crossprod(data.long$X)/n), useRaster=TRUE)
```



2 Implémentation des méthodes régularisées

On se propose d'étudier les procédures suivantes:

— la régression "stepwise" avec critère AIC et BIC,
— la régression "ridge",
— la régression "lasso",
— la variante "elastic-net" (ridge + lasso),

On comparera ces procédures dites régularisées aux méthodes de référence suivantes:

— les moindres carrés ordinaires,
— les moindres carrées oracle (où l'on suppose connaître $\mathcal{A}^\star$).

Écrire une fonction par estimateur, du type `getStepwiseAIC`, `getStepwiseBIC`, `getLasso`, etc. qui récupère la valeur de $\hat{\boldsymbol{\beta}}$. Toutes les méthodes seront implémentés à l'aide de la fonction `glmnet` du package du même nom sauf la régression stepwise (fonction `step` du package `MASS`).

Quelques remarques

— Pour les expériences en grande dimension ($n < p$), l'estimateur des moindres carrés n'est pas défini de manière unique: on utilisera un estimateur avec une faible pénalité ridge pour le régulariser.
— Pour les méthodes régularisées, on choisira le paramètre λ par validation croisée à l'aide des fonctions `cv.glmnet`.

```r
library(glmnet)
```

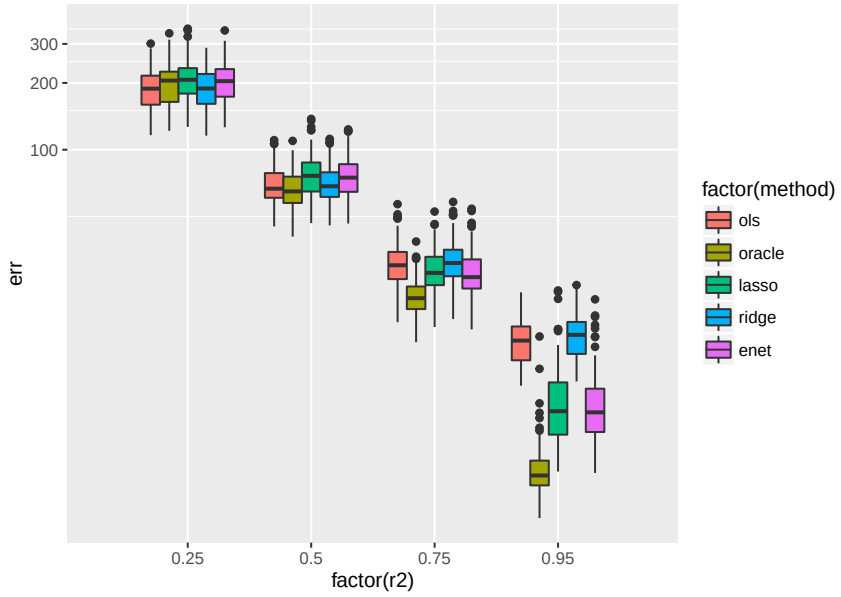
```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-5
```

```r
getLasso <- function(x,y) {
  out <- cv.glmnet(x,y,intercept=FALSE)
  return(predict(out$glmnet.fit, s = out$lambda.min, type="coefficient")[-1])
}

getRidge <- function(x,y) {
  out <- cv.glmnet(x,y,alpha=0,intercept=FALSE)
  return(predict(out$glmnet.fit, s = out$lambda.min, type="coefficient")[-1])
}

getEnet <- function(x,y) {
  out <- cv.glmnet(x,y,alpha=.5,intercept=FALSE)
  return(predict(out$glmnet.fit, s = out$lambda.min, type="coefficient")[-1])
}

getOLS <- function(x,y) {
  n <- nrow(x); p <- ncol(x)
  cv.out <- cv.glmnet(x,y,alpha=0, intercept=FALSE)
   if (n < p) {
    return(predict(cv.out$glmnet.fit, s = 1e-5, type="coefficient")[-1])
  } else {
    return(predict(cv.out$glmnet.fit, s = 0   , type="coefficient")[-1])
  }
}

getError <- function(beta,x.test,y.test) {
  return(mean((y.test - x.test %*% beta)^2))
}

data <- rlm(100, p=200, p0=10, r2 = 0.75)
x.train <- data$X[data$train, ]
y.train <- data$y[data$train]
x.test <- data$X[data$test, ]
y.test <-  data$y[data$test, ]
beta.lasso <- getLasso(x.train, y.train)
beta.ridge <- getRidge(x.train, y.train)
beta.enet  <- getEnet(x.train, y.train)
err <- c(lasso = getError(beta.lasso, x.test, y.test),
         ridge = getError(beta.ridge, x.test, y.test),
         enet  = getError(beta.enet , x.test, y.test))
```

Première simulations

```
library(reshape2)
n <- 50; p <- 100; p0 <- 10
rho <-.9
seq.r2 <- c(0.25, 0.5, 0.75, .95)
one.simu <- function(i) {
  out <- sapply(seq.r2, function(r2) {
   data <- rlm.long(n.train=n, p=p, p0=p0, rho = rho, r2 = r2)
   x.train <- data$X[data$train, ]
   y.train <- data$y[data$train]
   x.test <- data$X[data$test, ]
   y.test <-  data$y[data$test, ]
   A <- which(data$beta != 0)
   return(c(ols    = getError(getOLS(x.train, y.train), x.test, y.test),
            oracle= getError(getOLS(x.train[,A], y.train), x.test[,A], y.test),
            lasso = getError(getLasso(x.train, y.train), x.test, y.test),
            ridge = getError(getRidge(x.train, y.train), x.test, y.test),
            enet  = getError(getEnet(x.train, y.train) , x.test, y.test)))
   })
   colnames(out) <- seq.r2
   res <- melt(out)
   colnames(res) <- c("method", "r2", "err")
   res$simu <- i
   return(res)
}
nsim <- 100
library(parallel)
library(ggplot2)
res <- do.call(rbind, mclapply(1:nsim, one.simu, mc.cores =4))
ggplot(res, aes(x=factor(r2), y=err, fill=factor(method))) + geom_boxplot() + coord_tr
```



2.1 Lasso adaptatif

Le Lasso adaptatif est une version modifiée du lasso proposée pour palier notament au problème de biais. L'idée est de procéder en deux temps. Dans un premier temps, on estime des paramètres $\widehat{\beta}_{\text{init}}$ à l'aide du lasso. Ce premier estimateur va être utilisé comme poids pour une deuxième étape de lasso, en fixant $w_j = \widehat{\beta}_{j,\text{init}}$ tel que:

$$\hat{\boldsymbol{\beta}}_\lambda^{\text{ada}} = \underset{\boldsymbol{\beta}\in\mathbb{R}^p}{\arg\min} \left(\frac{1}{n}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \sum_{j=1}^{p} \frac{|\beta_j|}{|w_j|} \right).$$

L'idée est que si $\widehat{\beta}_{j,\text{init}} = 0$ alors $\widehat{\beta}_{j,\text{ada}} = 0$ de telle sorte que la première étape de lasso sert de pré-sélection. De plus, si $\widehat{\beta}_{j,\text{init}}$ est grand, le lasso adaptatif utilisera une pénalisation plus petite, donc un shrinkage plus petit pour le coefficient j, ce qui est sensé diminuer le bias pour ce coefficient.

Implémentez le lasso adaptatif à l'aide de l'option `penalty.factor` dans `glmnet` qui permet d'introduire des λs différents pour chaque élement de β. Créer une fonction `getadalasso(X,Y,beta.init)` qui calcule l'estimateur lasso adaptatif à partir d'un premier vecteur $\widehat{\beta}_{\text{init}}$.

2.1.1 Group-Lasso

Une autre modification populaire du Lasso est une version groupée du lasso – ou group-Lasso – qui suppose la connaissance d'une partition *a priori* des prédicteurs notées $\mathcal{G} = \{\mathcal{G}_1, \ldots, \mathcal{G}_K\}$. La sélection s'opère donc par groupe:

$$\hat{\boldsymbol{\beta}}_\lambda^{\text{grp}} = \underset{\boldsymbol{\beta}\in\mathbb{R}^p}{\arg\min} \left(\frac{1}{n}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \sum_{k=1}^{K} \|\boldsymbol{\beta}_{\mathcal{G}_k}\| \right).$$

Implémentez une fonction `getgrplasso` à l'aide du package `scoop`[1].

3 Comparaison des méthodes

3.1 Évaluation des performances

On s'intéresse aux performances des méthodes à la fois en terme de capacité prédictive et en terme de sélection de variables. On considère à cet égard les grandeurs suivantes:

- l'erreur quadratique moyenne de $\hat{\boldsymbol{\beta}}$ (`mse`)
- l'erreur moyenne de prédiction calculée sur l'ensemble test, (`err`)
- la précision du support estimé $\hat{\mathcal{A}}$ (`acc`, $(TN + TP)/p$)
- la sensibilité de $\hat{\mathcal{A}}$ (`spe`, $TP/(FN + TP)$)
- la spécificité de $\hat{\mathcal{A}}$ (`sen`, $TN/(TN + FP)$)

On a noté TP, FP, TN, FN respectivement pour *true positive*, *false positive*, *true negative* et *false negative*.

Écrire une fonction `getPerformance` qui calcule tous ces indices pour un estimateur $\hat{\boldsymbol{\beta}}$ donné.

3.2 Planning de simulations

Créer un script de simulation pour chaque scénario de matrice de covariance des prédicteurs $\boldsymbol{\Sigma}$, en commençant par exemple par le cas orthogonal.

Chaque simulation doit renvoyer un `data.frame` de la forme suivante, afin de faciliter le tracé des résultats à l'aide du package **ggplot2**.

```
##      method   mse   err  acc sen  spe n.p   r2 simu
## l1    lasso -0.72 -0.77 0.92 0.9 0.75 0.5 0.75    1
## l2    ridge  0.88  0.46 0.92 0.9 0.75 0.5 0.75    1
## l3 adalasso  1.41 -0.91 0.92 0.9 0.75 0.5 0.75    1
```

Écrire une fonction `one.simu(i)` permettant d'effectuer la simulation numéro i pour toutes les méthodes et pour toutes les valeurs des paramètres de simulation que vous aurez choisis (commencer doucement...). Cette fonction sera ensuite facilement parallélisable (par exemple avec le package **parallel**[2]).

3.2.1 Interprétations des résultats

- Représentez les boxplots des indicateurs de performance en fonction de n/p et de la valeur du R^2. On utilisera le package **ggplot2**.

1. à télécharger sur ma page web
2. attention: ne fonctionne pas sous Windows !

— Quels sont les effets de la grande dimension sur les performances des estimateurs (en estimation, en sélection) ?
— Explorer les différents scénarios. Y a t-il des méthodes plus adaptées à certains scénarios ? Si oui, pourquoi ?

3.3 Pour aller plus loin (et pour ceux qui sont en avance)

Cette partie a été rédigée par Franck Picard, merci à lui.

Nous nous interrogerons sur les conditions sur n, p et p_0 pour que le lasso détecte bien les entrées nulles et non-nulles de β^*. Dans un article publié en 2009 (IEEE Transactions on Information Theory, 55:2183–2202, May 2009), M. Wainwright propose des conditions nécessaires et suffisantes pour que le lasso soit consistant en sélection pour le support signé. On notera $\mathbb{S}_{\pm}(\beta)$ le vector de signes de β défini tel que:

$$\mathbb{S}_{\pm}(\beta_i) = \begin{cases} +1 & \text{si } \beta_i > 0 \\ -1 & \text{si } \beta_i < 0 \\ 0 & \text{si } \beta_i = 0 \end{cases}$$

Dans son article M. Wainwright démontre l'existence de deux constantes dépendant de $\Sigma = \mathbb{V}(X)$, $0 < \theta_\ell(\Sigma) \leq \theta_u(\Sigma) < \infty$ telles que pour une valeur

$$\lambda_n = \sqrt{\frac{2\sigma^2 \log(p_0)\log(p-p_0)}{n}}$$

du paramètre de régularisation du lasso,

— si $n/(2p_0(\log(p-p_0))) > \theta_u(\Sigma)$ alors il est toujours possible de trouver une valeur du paramètre de régularisation λ telle que le lasso a une solution unique $\widehat{\beta}$ telle que $\mathbb{P}\{\mathbb{S}_{\pm}(\beta^*) = \mathbb{S}_{\pm}(\widehat{\beta})\}$ tend vers 1.
— si $n/(2p_0(\log(p-p_0))) < \theta_\ell(\Sigma)$, alors quelle que soit la valeur du paramètre de régularisation $\lambda > 0$, aucune des solutions du lasso ne spécifie correctement le support signé de β^*, $\mathbb{P}\{\mathbb{S}_{\pm}(\beta^*) = \mathbb{S}_{\pm}(\widehat{\beta})\}$ tend vers 0.

Dans son article, M. Wainwright propose d'appeler la quantité $n/(2p_0(\log(p-p_0)))$ "taille d'échantillon normalisée". C'est un indicateur qui combine les informations nécessaires à la consistance du lasso. Dans la suite, nous nous placerons dans le cas $\Sigma = I$, avec $\theta_\ell(I) = \theta_u(I) = 1$.

3.3.1 Questions

— Pour les paramètres suivants, étudiez l'évolution de l'accuracy en terme de support en fonction de la taille d'échantillon normalisée, pour le lasso avec la valeur théorique de λ proposée ci-dessus. $p \in \{128, 256, 512\}$, $n \in \{100, \ldots, 1000\}$, $p_0 = \lceil 0.4 \times p \rceil$, $\beta_0^* = 0.5$, $\sigma = 0.5$.
— Comparer ces performances avec celles du lasso utilisant un λ calibré par validation croisée, et celles du lasso adaptatif (également avec λ calibré par validation croisée). Discutez les différences de comportement de l'accuracy.

4 Analyse de jeux de données omiques

4.1 Jeu de données "HIV"

4.1.1 Description

Jeu de données de génotypes associés au niveau du virus du VIH dans le sang. 605 individus ont été génotypés pour 300 SNPs.

4.1.2 Format

Lors du chargement des données, deux objets sont créés :

1. X - une matrice 605x300 donnant les génotypes de 605 individus pour 300 SNPs. 2. y - un vecteur de taille 605 donnant le niveau du virus du VIH dans le sang, pour chaque individu.

```
load("data/HIVdata.rda")
ls(); str(X); str(y)
```

```
## [1] "X" "y"
```

```
##  num [1:605, 1:300] 3 2 2 3 2 2 1 1 3 2 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:605] "030101" "030102" "060101" "060102" ...
##   ..$ : chr [1:300] "rs1264550" "rs9261947" "rs1079541" "rs9295871" ...
```

```
##  num [1:605] 4.56 5.18 5.2 6.18 6.11 4.64 2.91 6 5.9 4.74 ...
```

4.1.3 objectifs :

1. Sélectionner les SNPs qui sont associés au niveau du virus dans le sang.
2. Sélectionner des ensembles de SNPs intégrant une forme de structure de corrélation dans les données.
3. Évaluer les performances prédictives du modèles

4.1.4 référence

Dalmasso, C., Carpentier, W., Meyer, L., Rouzioux, C., Goujard, C., Chaix, M. L., ... & Theodorou, I. (2008). Distinct genetic loci control plasma HIV-RNA and cellular HIV-DNA levels in HIV-1 infection: the ANRS Genome Wide Association 01 study. *PloS one*, 3(12), e3907-e3907.

4.2 Jeu de données "colorectal"

4.2.1 Description

Jeu de données de niveaux d'expression de gène associés à des tissus tumoraux ou sain dans le cancer du colon. 62 tissus ont été analysés pour 2000 gènes ou assimilés.

4.2.2 Format

Lors du chargement des données, trois objets sont créés :

1. X - une matrice 62x2000 donnant les niveaux d'expression (log tranformés) relevé dans les tissus du colons de 62 patients.
2. y - un vecteur de taille 62 indiquant le statut du tissus (-1: tumoral, 1: sain).
3. genes.info - une liste de longueur 2000 donnant des informations sur les 2000 gènes considérés.

```
load("data/colorectal.rda")
ls()
```

```
## [1] "genes.info" "X"          "y"
```

4.2.3 objectifs :

1. Sélectionner les gènes liés au cancer colo-rectal à l'aide d'un modèle gaussien
2. Sélectionner les gènes liés au cancer colo-rectal à l'aide d'un modèle logistique.
3. Prédire le statut d'un tissu.

4.2.4 référence

U. Alon, N. Barkai, D. A. Notterman, K. Gish, S. Ybarra, D. Mack, and A. J. Levine, "Broad patterns of gene expression revealed by clustering of tumor and normal colon tissues probed by oligonucleotide arrays", *PNAS*, vol. 96, 1999.

4.3 Jeu de données 'Bardet'

4.3.1 description

Jeu de données simplifié d'expression de gènes associées au syndrome de Bardet-Biedl. Les échantillons ont été biopsiés à partir de tissus d'oeil de 120 rats.

4.3.2 format

Lors du chargement des données, la liste `bardet` est créé, contenant deux objets:

1. x - une matrice 120 x 100, donnant les expressions associées à 120 rats pour 100 sondes associés à 20 gènes. 5 sondes consécutives correspondent au même gène.
2. y - un vecteur de taille 120 donnant le niveau d'expression du gène TRIM32.

```
load("data/bardet.rda")
str(bardet)
```

```
## List of 2
##  $ x: num [1:120, 1:100] 0.44705 0.46684 0.01498 0 0.00281 ...
##  $ y: num [1:120] 8.42 8.36 8.41 8.29 8.27 ...
```

4.3.3 objectifs

1. Sélectionner les sondes les plus prédictives de l'expression de TRIM32.
2. Opérer une sélection de sonde "par groupe" associée à chaque gène, sachant que 5 prédicteurs consécutifs sont des sondes associées au même gène.
3. Évaluer les performances prédictives du modèle

4.3.4 référence

T. Scheetz, K. Kim, R. Swiderski, A. Philp, T. Braun, K. Knudtson, A. Dorrance, G. DiBona, J. Huang, T. Casavant, V. Sheffield, E. Stone .Regulation of gene expression in the mammalian eye and its relevance to eye disease. *Proceedings of the National Academy of Sciences of the United States of America*, 2006.