Projet de statisques pour la Genomique

Naila Bouterfa

5 mai 2018

Introduction

Nous choisissons une base de donnees sur le cancer des ovaires “GSE14764” dans la librarie “Curated Ovarian Data” et faisons l’extraction des variables expression de genes et d’une variable qualitative binaire qu’on va chercher a predire apres avoir tout en definissant la signature associee. Cette variable ici Y est celle relative a la recurrence de la maladie. Elle prend la veleur “1” si il y a récurrence du cancer et “0” si il n’y en a pas. Nous allons mener pour ca une regression lineaire penalisee sur ses donnees avec etude de la stabilite du modele puis dans un second temps une regression PLS parcimonieuse dont ont analysera egalement la stabilite.

Importation de donnees

Dans ce qui suit nous importons les donnees d’expression de genes d’un cote et la donnees phenotypique que nous traduisons en binaire de l’autre. Le traitement necessite aussi de transposer la matrice d’expression de genes et de supprimer les valeurs manquantes. Au final nous avons une matrice avec 76 individus et 13104 genes.

```
rm(list=ls())
source("https://bioconductor.org/biocLite.R")
#biocLite("curatedOvarianData")
library("curatedOvarianData")
#biocLite("Biobase")
library('Biobase')

data(GSE14764_eset)

expressionData <- exprs(GSE14764_eset)

otherData <- pData(GSE14764_eset@phenoData)

Y <- (otherData$recurrence_status == "recurrence")*1

M=t(expressionData)

M=M[-which(is.na(Y)),]
Y=Y[-which(is.na(Y))]
```

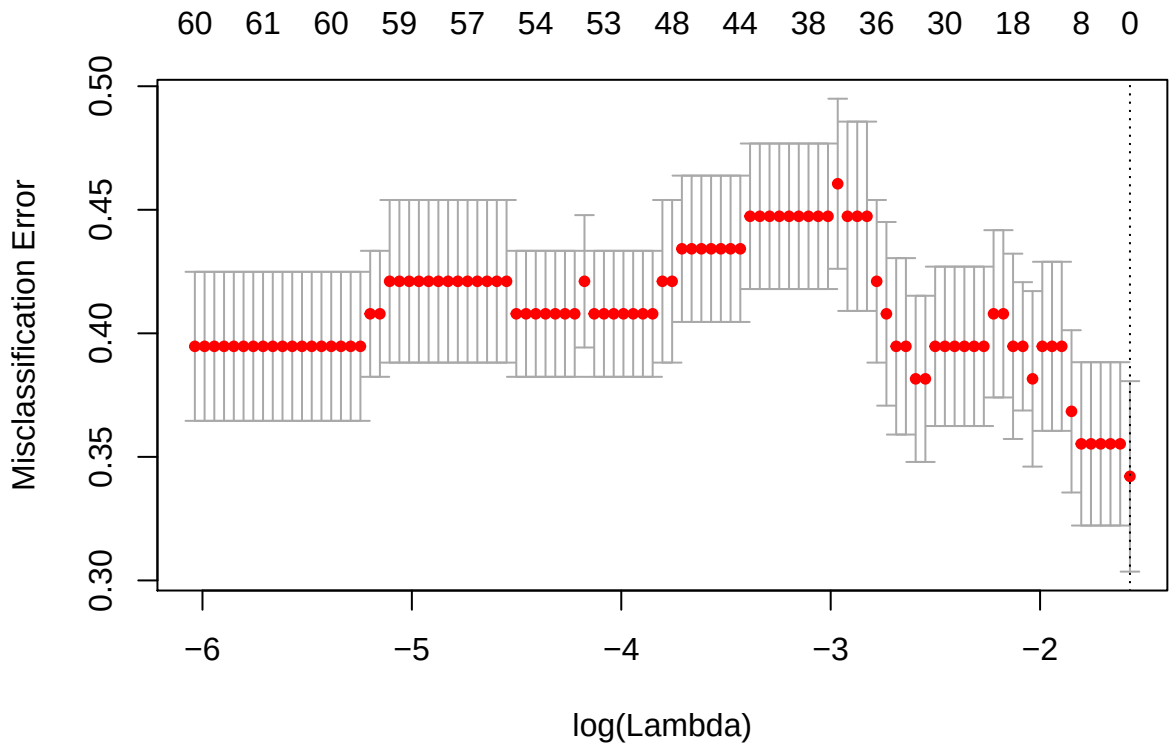
Regression penalisee

Nous procedons dans un premier temps a une validation croisee qui nous servira pour extraire la valeur de lambda minimale puis nous procedons a une regression penalisee avec cette valeur optimale sur un echantillon que nous divisons en apprentissage et echantillon test. Nous procedons enfin a une prediction dont nous calculons le taux erreur et le pourcentage d’erreur associe.

```
library(glmnet)
```

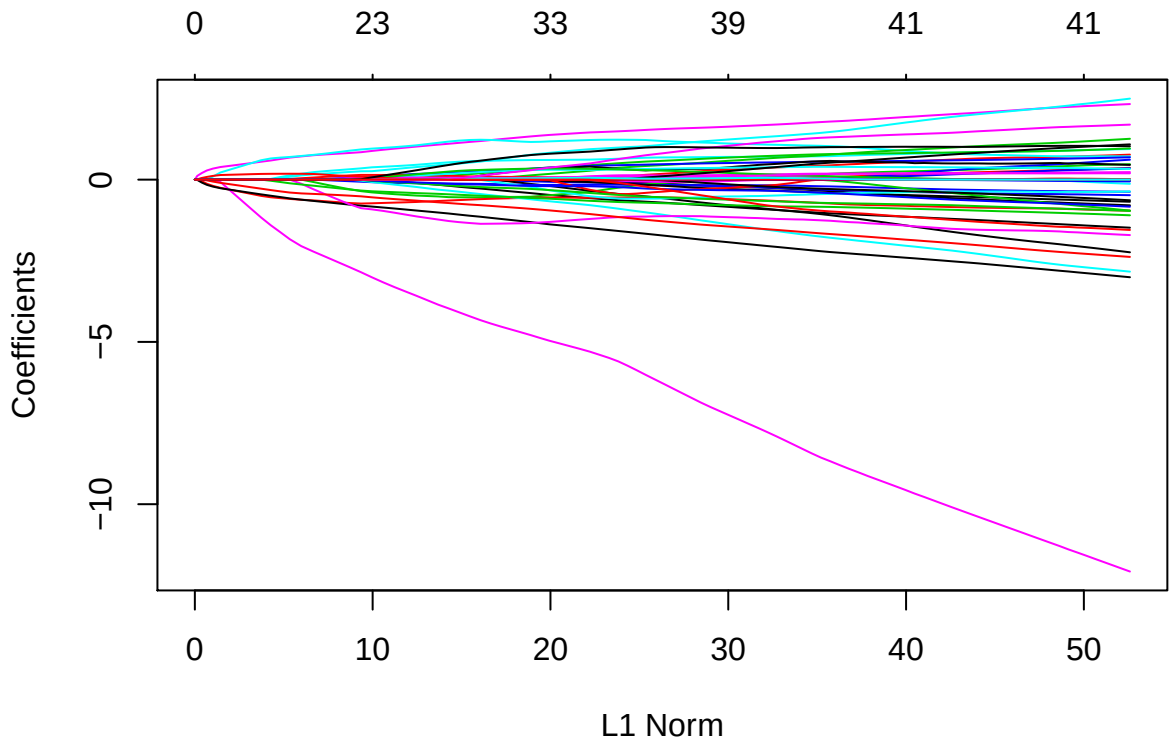
```
## Warning: package 'glmnet' was built under R version 3.4.4
## Loading required package: Matrix
## Loading required package: foreach
## Warning: package 'foreach' was built under R version 3.4.4
## Loaded glmnet 2.0-16
```

```
RegCv = cv.glmnet(M,Y,family='binomial',type.measure = "class")

Lmin=RegCv$lambda.min

plot(RegCv)
```




```
ind=sample(1:76)

indtrain=ind[1:60]
indtest=ind[60:76]

Mtrain=M[indtrain,]
Ytrain=Y[indtrain]

Mtest=M[indtest,]
Ytest=Y[indtest]
```

```
Reg=glmnet(Mtrain,Ytrain,family="binomial")

plot(Reg)
```




```
coefficients1=coef(Reg,s=Lmin)

prediction1 = predict(Reg,Mtest,s=Lmin,type="class")

Taux_Err1=sum(Ytest!=prediction1)/length(Ytest)
Taux_Err1
```

```
## [1] 0.2941176
```

```
Pourc_err1=Taux_Err1*100
Pourc_err1
```

```
## [1] 29.41176
```

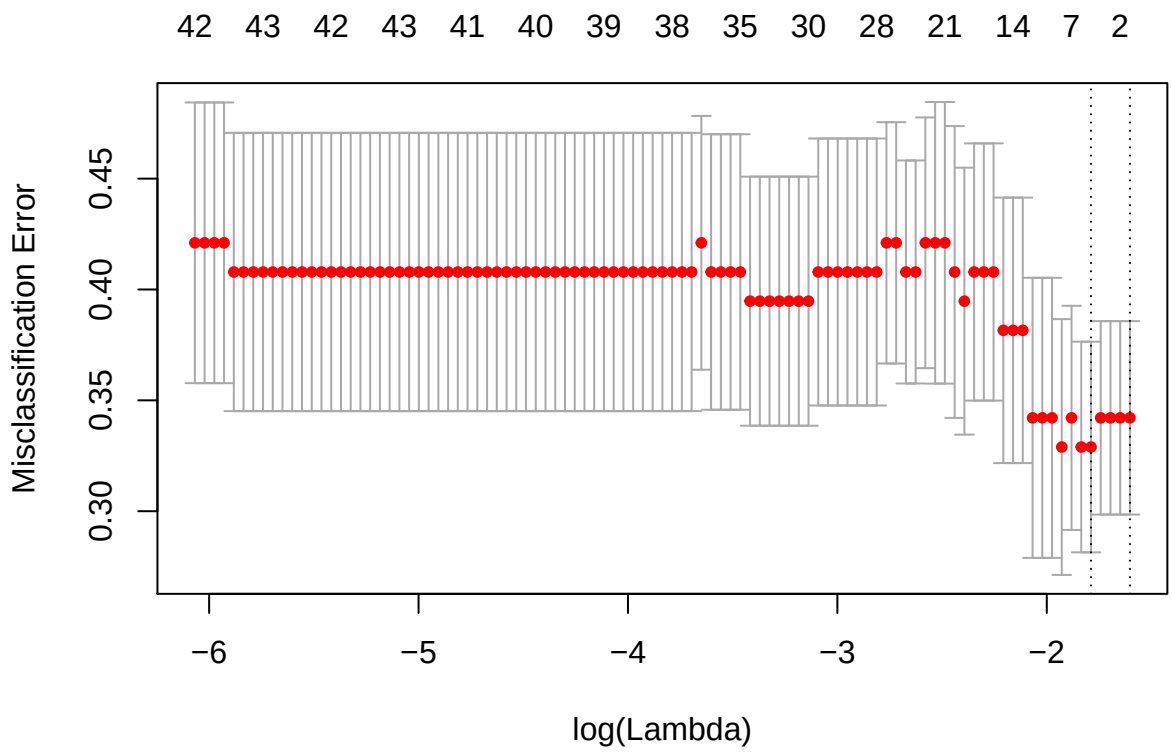
Etude de stabilte de la regression lineaire penalisee

Construction de la fonction bootstrap :

```
bootstrap = function (Ma){
mix=sample(1:dim(Ma)[1],dim(Ma)[1],replace=TRUE)
Ma[mix,]
}
```

Boostrap sur notre matrice de donnees :

```
Mnew=bootstrap(M)

RegCv2 = cv.glmnet(Mnew,Y,family='binomial',type.measure = "class")

Lmin=RegCv2$lambda.min

plot(RegCv2)
```




```
ind=sample(1:76)

indtrain=ind[1:60]
indtest=ind[60:76]

Mtrain=Mnew[indtrain,]
Ytrain=Y[indtrain]

Mtest=Mnew[indtest,]
Ytest=Y[indtest]

Reg2=glmnet(Mtrain,Ytrain,family="binomial")

coefficients2=coef(Reg2,s=Lmin)

prediction2 = predict(Reg2,Mtest,s=Lmin,type="class")
```

```
Taux_Err2=sum(Ytest!=prediction2)/length(Ytest)
Taux_Err2
```

```
## [1] 0.1764706
```

```
Pourc_err2=Taux_Err2*100
Pourc_err2
```

```
## [1] 17.64706
```

Commentaire : En effectuant un bootstrap on obtient un taux d'erreur different ce qui prouve l'instabilite du modele.

On peut maintenant effectuer plusieurs bootsrap comme il suit :

```
R=100
Coeff=list()

for (j in 1:R){
  Mnew=bootstrap(M)
  cvfit = cv.glmnet(Mnew,Y,family='binomial',type.measure = "class")
  Lmin=cvfit$lambda.min
  reg=glmnet(M,Y,family="binomial")
  co=coef(reg,s=Lmin)
  Coeff[[j]]=co@i
}

b = unlist(Coeff)
sort(table(b),decreasing = TRUE)
```

```
## b
##     0   938 12162  4056 11927  2650  5474   421 10139  9778  9896  8580
##   100    91    83    69    61    60    60    53    34    30    30    23
##  2811 11003  6155  9936  6929  2568  7771  8660  8322  8964  7799 11177
##    22    22    21    21    20    17    17    17    11    11     9     9
## 13003   140   536  1982  9943 12304  1175  3482  1415  3444  6392 11519
##     9     8     8     8     8     8     7     7     6     6     6     5
## 12072    29   374  1059  6418  9093  9415   990  1161  2833  3343  5478
##     5     4     4     4     4     4     4     3     3     3     3     3
##  1349  2712  3120  5591  6145 12430   608   777  2176  3501  4005 10283
##     2     2     2     2     2     2     1     1     1     1     1     1
## 10701 12671
##     1     1
```

```
num_selection_signature1=sort(table(b),decreasing = TRUE)[2:7]
num_selection_signature1
```

```
## b
##   938 12162  4056 11927  2650  5474
##    91    83    69    61    60    60
```

```
names_selection_signature1=colnames(M[,as.numeric(names(num_selection_signature1))])
names_selection_signature1
```

```
## [1] "ATRIP///TREX1" "TUSC3"         "FRMD1"         "TREM2"
## [5] "CX3CR1"        "INSIG1"
```

On choisit de retenir les genes qui apparaissent un grand nombre de fois. Ces genes sont classes dans un

ordre decroissant de pourcentage d'apparition.

Limites de la methode LASSO :

- Que peut-on reprocher a cette methode en termes de prise en compte de la colinearité?

On obtient de bon estimateurs cependant la methode ne traite pas le probleme de correlation des variables et la variance des estimateurs reste elevee. En effet cette methode privilegiera une variable au detriment des autres du fait d'une forte correlation entre les deux. On perd donc en information.

- Quelles en sont les conséquences possibles sur la signature ?

On peut selectionner une mauvais variable qui est tres correlee a la bonne variable. Ou alors une seule parmi plusieurs variables interessantes d'ou la perte d'information.

- Voyez-vous des pistes a explorer pour ameliorer cela?

Les methodes "Elastic net" qui ajoute une penalite ridge au Lasso ou la methode "Group Lasso" qui fournit des groupes de genes au lieu de genes peuvent donner de meilleurs resultats.

Evaluation du modele construit sur la signature :

```
Msign=M[,names_selection_signature1]

ind=sample(1:76)

indtrain=ind[1:60]
indtest=ind[60:76]

Mstrain=Msign[indtrain,]
Ytrain=Y[indtrain]

Mstest=Msign[indtest,]
Ytest=Y[indtest]

RegScv=cv.glmnet(Msign,Y,family="binomial",type.measure = "class")
lmin=RegScv$lambda.min

RegS=glmnet(Mstrain,Ytrain,family="binomial")

prediction3=predict(RegS,Mstest,s=lmin,type="class")

Taux_Err3=sum(Ytest!=prediction3)/length(Ytest)
Taux_Err3
```

```
## [1] 0
```

```
Pourc_err3=Taux_Err3*100
Pourc_err3
```

```
## [1] 0
```

Le taux d'erreur est globalement reduit. On a stabilise le modele de cette facon.

Regression PLS parcimonieuse

```
library("plsgenomics")
```

```
lambdas=seq(0,1,0.1)

sparsePLScv=spls.cv(X=M,Y=Y,lambda.l1.range=lambdas, ncomp.range=1:20)

#valeurs optimales :

lambda=sparsePLScv$lambda.l1.opt
comp=sparsePLScv$ncomp.opt

ind=sample(1:76)

indtrain=ind[1:60]
indtest=ind[60:76]

MTrain=M[indtrain,]
YTrain=Y[indtrain]

MTest=M[indtest,]
YTest=Y[indtest]


sparsePLS=spls(Xtrain=MTrain,Ytrain=YTrain,lambda.l1 = lambda,ncomp=comp,Xtest=MTest)
```

Etude de stabilite de la methode PLS parcimonieuse

```
R=100
Coeff=list()

for (j in 1:R){
  Mnew=bootstrap(M)
  MTest=Mnew[indtest,]
  MTrain=Mnew[indtrain,]
  sparsePLS=spls(Xtrain=MTrain,Ytrain=YTrain,lambda.l1 = lambda,ncomp=comp,Xtest=MTest)
  Coeff[[j]]=sparsePLS$A
}

c = unlist(Coeff)

num_selection_signature2=sort(table(c),decreasing = TRUE)[2:7]
num_selection_signature2
```

```
## c
##  8016 10021  3227  5620  8911  9205
##    65    65    64    64    64    64
```

```
names_selection_signature2=colnames(M[,as.numeric(names(num_selection_signature2))])
names_selection_signature2
```

```
## [1] "PBX2"    "SCTR"    "DYRK1B"  "JPH2"    "PRSS3P3" "RABEP2"
```