TP2-KLR-Rapport

Antoine Bourgoin & Benoit Charmettant

Pour le 18/11/2020

```
library(glmnetUtils)
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.0-2
```

```
##
## Attaching package: 'glmnet'
```

```
## The following objects are masked from 'package:glmnetUtils':
##
##     cv.glmnet, glmnet
```

```
ad_dataset <- read.table("Alzheimer_Webster.txt")
```

Cas pratique

Commençons par charger les données. Et séparer les paramètres des labels

```
X <- model.matrix(Y~., data=ad_dataset)[,-1]
Y <- ad_dataset$Y
dim(X)
```

```
## [1]  364 8650
```

```
sum(Y)
```

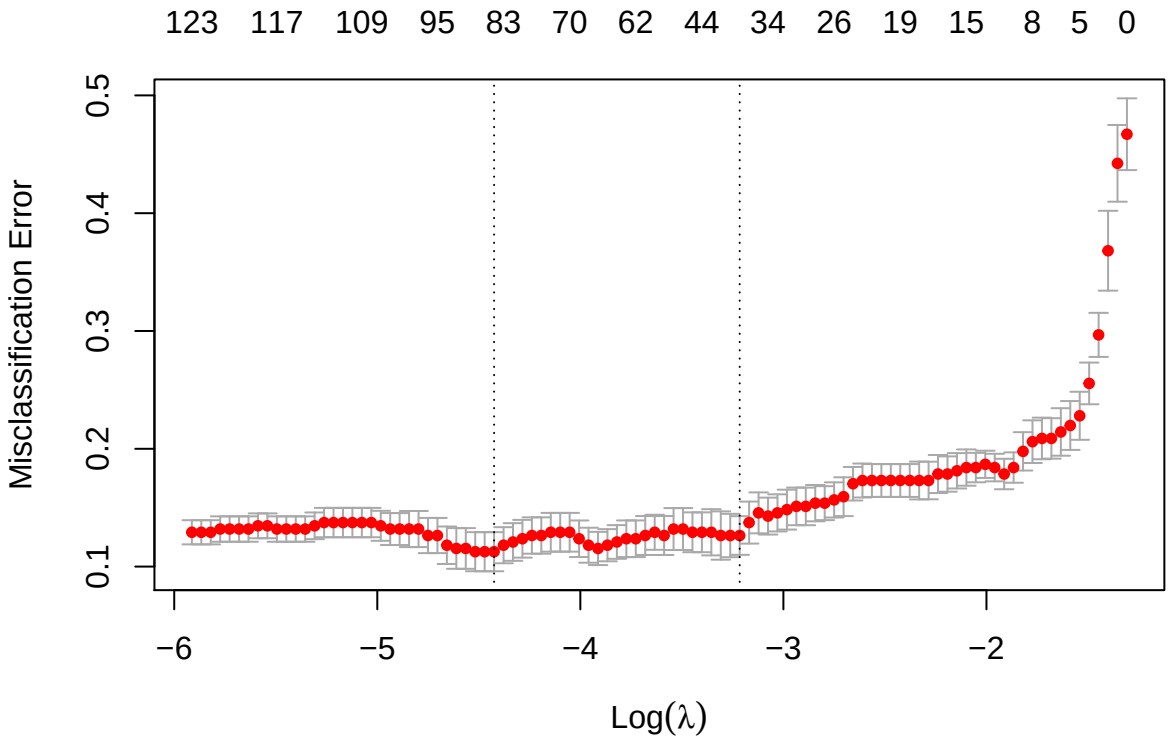
```
## [1] 176
```

Le set de données est composé de 364 patients au total pour lesquels sont disponible l'expression de 8650 gène. Y regroupe les labels de chaque patient (0 pour les 188 patients du groupe de contrôle, non affecté par la maladie d'alzheimer et 1 pour les 176 patients atteints par cette maladie)

Question 1

Le jeu de données étant de grande dimension (8650 gènes par patient) nous allons utiliser une régression régularisé par méthode Lasso ($\alpha = 1$) ce qui permet de d'éliminer un grand nombre de dimensions (coefficient nul). La fonction glmnet permet d'optimiser un modèle de régression logistique via l'algorithme de Newton.

```
fit.data <- cv.glmnet(X, Y, alpha = 1, family = "binomial", type.measure ="class")
plot(fit.data)
```



Taux d'erreur selon la valeur du paramètre lambda pour une régularisation Lasso

```
head(coef(fit.data))
```

```
## 6 x 1 sparse Matrix of class "dgCMatrix"
##                       1
## (Intercept)   1.811166
## GI_10047091.S .
## GI_10047093.S .
## GI_10047103.S .
## GI_10047133.A .
## GI_10092596.S .
```

Si l'on affiche les premier coefficients on s'aperçoit qu'ils sont effectivement nuls pour un majorité d'entre eux.

Nous pouvons alors récupérer les gènes pour lesquels les coefficients ne s'annulent pas.

```
library(coefplot)
head(extract.coef(fit.data))
```

```
##                       Value SE   Coefficient
## (Intercept)   -2.0331569743 NA   (Intercept)
## GI_10835100.S -0.0054612838 NA GI_10835100.S
## GI_13259500.A  0.0117547785 NA GI_13259500.A
## GI_14149858.S  0.0065805963 NA GI_14149858.S
## GI_14249703.S -0.0097166832 NA GI_14249703.S
## GI_15451905.S -0.0008929511 NA GI_15451905.S
```

```
dim(extract.coef(fit.data))
```

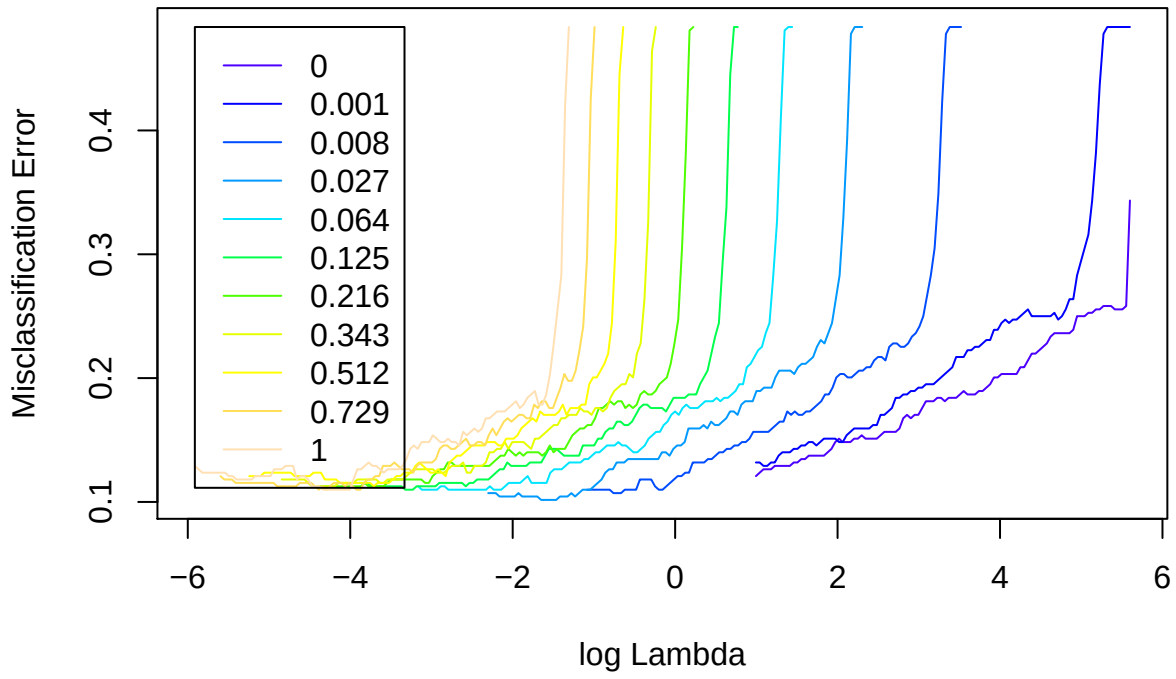
```
## [1] 84  3
```

Sur les 8650 gènes seuls 68 ont été sélectionnés par le modèle, pour un taux d'erreur proche de 10%.

Question 2

Nous pouvons maintenant considérer une regression elastic net et faire varier le paramètre α qui répartie la régularisation entre une régularisation Lasso et Ridge ainsi que le paramètre λ qui fait varier le poids de la régularisation lors de l'optimisation.

```
fit.data.b <- cva.glmnet(X, Y, family = "binomial", type.measure ="class")
plot(fit.data.b)
```



Taux d'erreur selon la valeur de α (pour chaque courbe de couleur) et $l\lambda$ abscisse.

```
# Get alpha.
get_alpha <- function(fit) {
  alpha <- fit$alpha
  error <- sapply(fit$modlist, function(mod) {min(mod$cvm)})
  alpha[which.min(error)]
}

# Get all parameters.
get_model_params <- function(fit) {
  alpha <- fit$alpha
  lambdaMin <- sapply(fit$modlist, `[[`, "lambda.min")
  lambdaSE <- sapply(fit$modlist, `[[`, "lambda.1se")
  error <- sapply(fit$modlist, function(mod) {min(mod$cvm)})
  best <- which.min(error)
```

```
  data.frame(alpha = alpha[best], lambdaMin = lambdaMin[best],
             lambdaSE = lambdaSE[best], eror = error[best])
}

get_model_params (fit.data.b)
```

```
##   alpha lambdaMin  lambdaSE      eror
## 1 0.027 0.2312996 0.3858311 0.1016484
```

Le meilleur taux de classification est obtenu pour ces paramètres

On remarque que le α optimal ne se situe pas très proche de 1 commme supposé lors de la question précédente mais c'est car les résultats sont très proches pour $\alpha > 0.1$. Sa valeur minimale est très surement due à la fluctuation statistique.