Kernel Logistic Regression

Antoine Dargier

2022-11-05

1. Introduction

Une population est divisée en 2 classes au moyen d'un critère qualitatif Y . Chaque individu de la population est décrit par p variables $X1, ..., Xp$. La régression logistique est une méthode statistique adaptée à l'étude de la liaison entre la variable qualitative Y les p variables explicatives $X1, ..., Xp$. Ici, nous nous intéressons à la régression logistique régularisée à noyau

2. La régression logistique binaire multiple

Question 1. Écrire la vraisemblance $L(\beta)$ du modèle

$$L(\beta) = \prod_{i=1}^{n} P(Y = y_i | X = x_i)$$

Si $y_i = 1$

$$P(Y = y_i | X = x_i) = \pi(x_i)$$

Si $y_i = 0$

$$P(Y = y_i | X = x_i) = 1 - \pi(x_i)$$

$$DoncL(\beta) = \prod_{i=1}^{n} \pi(x_i)^{y_i} (1 - \pi(x_i))^{1-y_i}$$

Question 2. Montrer que l'estimateur du maximum de vraisemblance peut s'obtenir en considérant l'algorithme itératif suivant :

$$\beta^{(s+1)} = \beta^{(s)} + (X^T V^{(s)} X)^{-1} X^T (y - \pi^{(s)})$$

où :

- X est la matrice formée d'une première colonne de coordonnées constantes égales à 1 et des p colonnes correspondant aux variables $X1, ..., Xp$ observées sur les n individus.
- $y = (y_1, ..., y_n)^T$ est le vecteur colonne de labels associés à chaque x_i
- $\pi^{(s)}$ est le vecteur formé des $\pi_i = \pi(x_i)$ estimé à l'itération courante s
- $V^{(s)}$ est la matrice diagonale formée des $\pi_i^{(s)}(1 - \pi_i^{(s)})$

Indications : On considérera le développement de Taylor de la log-vraisemblance $L(\beta) = \log(L(\beta))$ à l'ordre 2 en $\beta^{(s)}$

$$L(\beta) = \ln(L(\beta)) = \sum_{i=1}^{n} y_i \ln(x_i) + (1-y_i)\ln(1-\pi(x_i)) = \sum_{i=1}^{n} (y_i \ln(\frac{\pi_i}{1-\pi_i}) + \ln(1-\pi_i)) = \sum_{i=1}^{n} (y_i x_i^T \beta - \ln(1+\exp^{x_i^T \beta}))$$

Pour obtenir le maximum de vraisemblance, nous allons dériver $L(\beta)$ par rapport aux β_i et égaliser à 0.

Nous obtenons :

$$U = \begin{pmatrix} \frac{\partial L}{\partial \beta_0} \\ \vdots \\ \frac{\partial L}{\partial \beta_j} \end{pmatrix} = 0$$

Or,

$$\frac{\partial L}{\partial \beta_j} = \sum_{i=1}^{n} y_i x_{ij} - x_{ij} \frac{e^{x_i^T \beta}}{1 + e^{x_i^T \beta}} = \sum_{i=1}^{n} y_i x_{ij} - x_{ij} \pi_i$$

Donc nous avons : $U = X^T(y - \pi) = 0$, avec

$$X = \begin{pmatrix} \cdots \\ x_i^T \\ \cdots \end{pmatrix}$$

$$\pi = \begin{pmatrix} \pi_1 \\ \vdots \\ \pi_n \end{pmatrix}$$

$$y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

Pour résoudre ce problème d'optimsation, nous allons utiliser l'algorithme de Newton-Raphson, qui permet de trouver le zéro d'une fonction.

En utilisant le développement de Taylor à l'ordre 2 d'une fonction f, nous avons : $f(a+h) \approx f(a) + h^T \nabla f(a) + \frac{1}{2} h^T H h$

où H est la matrice hessienne des dérivées secondes :

$$[H]_{jk} = \frac{\partial L}{\partial \beta_j \partial \beta_k} = \frac{\partial \sum_{i=1}^{n} y_i x_{ij} - x_{ij}\pi_i}{\partial \beta_k} = -\sum_{i=1}^{n} x_{ij} \frac{\partial \pi_i}{\partial \beta_k} = -\sum_{i=1}^{n} x_{ij} x_{ik} \pi_i (1-\pi_i) = -X^T V X$$

Dans la formule de Taylor, nous pouvons donc choisir $a = \beta^{(s)}$ et $h = \beta - \beta^{(s)}$.

Nous obtenons donc :

$$L(\beta) = L(\beta^{(s)}) + (\beta - \beta^{(s)})^T U(\beta^{(s)}) + \frac{1}{2}(\beta - \beta^{(s)})^T H(\beta^{(s)})(\beta - \beta^{(s)})$$

Nous dérivons cette expression par rapport à β, et nous choisissons $\beta^{(s+1)}$ qui annule cette dérivée :

$$0 = U(\beta^s) + H(\beta^s)(\beta^{s+1} - \beta^s) \Rightarrow \beta^{(s+1)} = \beta^{(s)} - H(\beta^{(s)})^{-1}U(\beta^{(s)}) = \beta^{(s)} + (X^T V^{(s)} X)^{-1} X^T (y - \pi^{(s)})$$

Nous obtenons bien l'équation voulue. De plus, H étant définie négative, le maximum de vraisemblance est une fonction concave, donc le maximum est atteint.

Question 3. En remarquant que $\pi_i(1-\pi_i)$ est majorée par $\frac{1}{4}$, proposer une approximation H_2 de la matrice $H_1 = X^T V^{(s)} X$ telle que $H_1 - H_2$ soit définie positive.

Soit

$$f : x \mapsto x(1-x) f'(x) = 1 - 2x = 0 \Rightarrow x = \frac{1}{2} et f(x*) = \frac{1}{4} Donc f \leq \frac{1}{4}$$

Soit $H_2 = -\frac{1}{4} X^T X$, nous avons :

$$H_1 - H_2 = -X^T \begin{pmatrix} \pi_1(1-\pi_1) - \frac{1}{4} & 0 & \cdots \\ 0 & \ddots & 0 \\ 0 & \cdots & \pi_n(1-\pi_n) - \frac{1}{4} \end{pmatrix} X$$

Et, comme $\pi_i(1-\pi_i) \leq \frac{1}{4}$, nous avons bien $H_1 - H_2$ une matrice définie positive.

Question 4. Réécrire l'algorithme itératif en y injectant cette approximation.

Nous avons désormais :

$$\beta^{(s+1)} = \beta^{(s)} + (\frac{1}{4} X^T X)^{-1} X^T (y - \pi^{(s)}) = \beta^{(s)} + 4(X^T X)^{-1} X^T (y - \pi^{(s)})$$

Question 5. Discuter l'intérêt de considérer la maximisation de la vraisemblance pénalisée.

On considère maintenant la vraisemblance pénalisée définie comme suit :

$L_\lambda(\beta) = L(\beta) - \frac{\lambda}{2} \|\beta_\lambda\|_2^2$

où le paramètre λ est un paramètre de régularisation.

Cette pénalisation permet un meilleur contrôle biais/variance, et donc d'éviter l'overfitting.

Question 6. Montrer que la maximisation de la vraisemblance pénalisée peut s'obtenir en considérant l'algorithme itératif suivant :

$$\beta_\lambda^{(s+1)} = \beta_\lambda^{(s)} + 4(X^T X + 4\lambda I_p)^{-1} (X^T (y - \pi) - \lambda \beta_\lambda^{(s)})$$

En reprenant le raisonnement de la question 2, nous avons désormais :

$$U = \frac{\partial L - \frac{\lambda}{2} \|\beta_\lambda\|_2^2}{\partial \beta_j} = \sum_{i=1}^{n} y_i x_{ij} - x_{ij} \pi_i - \lambda \beta_{\lambda j} Donc, U = X^T (y - \pi) - \lambda \beta_\lambda$$

De même, $H = -X^T V X + \lambda I_p$ En reprenant les mêmes notations que dans les questions précédentes, nous avons désormais : $H_1 = X^T V X - \lambda I_p$, et cette fois $H_2 = -\frac{1}{4} X^T X - \lambda I_p$. Nous avons bien $H_1 - H_2$ définie positive, et donc

$$H_2^{-1} = (-\frac{1}{4} X^T X - \lambda I_p)^{-1} = -4(X^T X + 4\lambda I_p)^{-1}$$

Finalement, en réinsérant ces nouvelles expressions de U et H dans la formule de l'algorithme de Newton-Raphson, nous obtenons :

$$\beta_\lambda^{(s+1)} = \beta_\lambda^{(s)} - H(\beta_\lambda^{(s)})^{-1}U(\beta_\lambda^{(s)}) = \beta_\lambda^{(s)} + 4(X^TX + 4\lambda I_p)^{-1}(X^T(y-\pi) - \lambda\beta_\lambda)$$

Question 7. Montrer que la maximisation de la vraisemblance pénalisée revient à résoudre le problème problème d'optimisation suivant :

$$\min_\beta \sum_{i=1}^{n} log(1 + exp^{-\tilde{y_i}\beta^T x_i}) + \frac{\lambda}{2}\left\|\beta_\lambda\right\|_2^2$$

où $\tilde{y_i}$ est la variable à prédire encodée en {-1, 1} (-1 au lieu de 0)

Considérons $\tilde{y_i}$ la variable à prédire encodée en {-1, 1} (-1 au lieu de 0).

Si $\tilde{y_i} = 1$, $L(\beta) = \pi(x_i) = \frac{exp^{\beta^T x}}{1+exp^{\beta^T x}} = \frac{1}{1+exp^{-\beta^T x}} = \frac{1}{1+exp^{-\tilde{y_i}\beta^T x}}$

Si $\tilde{y_i} = -1$, $L(\beta) = 1 - \pi(x_i) = \frac{1}{1+exp^{\beta^T x}} = \frac{1}{1+exp^{-\tilde{y_i}\beta^T x}}$

Nous pouvons donc réécrire le problème de maximisation de la vraisemblance pénalisée sous la forme du problème d'optimisation suivant :

$$\max_\beta \sum_{i=1}^{n} log(\frac{1}{1+exp^{-\tilde{y_i}\beta^T x_i}}) - \frac{\lambda}{2}\left\|\beta_\lambda\right\|_2^2 \Leftrightarrow \max_\beta \sum_{i=1}^{n} -log(1+exp^{-\tilde{y_i}\beta^T x_i}) - \frac{\lambda}{2}\left\|\beta_\lambda\right\|_2^2 \Leftrightarrow \min_\beta \sum_{i=1}^{n} log(1+exp^{-\tilde{y_i}\beta^T x_i}) + \frac{\lambda}{2}\left\|\beta_\lambda\right\|_2^2$$

Question 8. En supposant que XX^T est de rang plein, montrer qu'une version duale de l'algorithme de régression logistique régularisée est définie par :

$\alpha_\lambda^{(s+1)} = \alpha_\lambda^{(s)} + 4(XX^T + 4\lambda I_n)^{-1}(y - \pi - \lambda\alpha_\lambda^{(s)})$

A l'issue de la question 7, on applique le théorème du représentant qui stipule que $\beta = X^T\alpha = \sum_{i=1}^{n}\alpha_i x_i$.

Donc, l'équation obtenue à la question 6 donne :

$$X^T\alpha_\lambda^{(s+1)} = X^T\alpha_\lambda^{(s)} + 4(X^TX + 4\lambda I_p)^{-1}(X^T(y-\pi) - \lambda X^T\alpha_\lambda^{(s)}) = X^T\alpha_\lambda^{(s)} + 4(X^TX + 4\lambda I_p)^{-1}X^T(y - \pi - \lambda\alpha_\lambda^{(s)})$$

Or, $(X^TX + \lambda I_p)^{-1}X^T = X^T(XX^T + \lambda I_p)^{-1}$

Alors,

$$X^T\alpha_\lambda^{(s+1)} = X^T\alpha_\lambda^{(s)} + 4X^T(XX^T + \lambda I_p)^{-1}(y - \pi - \lambda\alpha_\lambda^{(s)})$$

On multiplie à les deux termes à gauche par X, et nous pouvons simplifier par XX^T, car XX^T est inversible (de rang plein). On obtient alors :

$\alpha_\lambda^{(s+1)} = \alpha_\lambda^{(s)} + 4(XX^T + 4\lambda I_n)^{-1}(y - \pi - \lambda\alpha_\lambda^{(s)})$

3. Cas pratique - Résolution avec glmnet

J'ai choisi dans cette partie pratique de travailler sur les données Alzheimer. Dans un premier temps, nous allons chercher à prédire le statut des patients en utilisant une régression logistique et la librairie glmnet. Cela va nous permettre de découvrir les données, d'avoir des premiers intuitions de résultats, ... Nous verrons dans la partie 4 l'implémentation de l'algorithme de Newton-Raphson et des équations vues dans la partie précédente.

```
library(glmnet)
library(dplyr)
library(pracma)
library(pROC)
library(caret)
```

Nous commençons notre étude par une brève visualisation des données.

```
data = read.table("C:\\Users\\antoi\\OneDrive\\Bureau\\CS\\3A\\SDI\\ML\\Cours 4\\Alzheimer_Webster.txt")
head(data)
```

Après une première exploration des données, nous pouvons constater que nous avons une liste de 364 patients, avec pour chacun l'expression de 8650 gènes. La dernière colonne, nommée Y, vaut 0 pour les patients non atteints, et 1 pour les patients atteints par Alzheimer. L'objectif de cette étude va être d'entrainer un modèle pour prédire le statut du patient (atteint ou non) en fonction de l'expression de ses gènes.

Nous allons commencer par diviser nos données en deux, pour créer deux ensembles de données : les données d'entrainement sur lesquelles nous allons entrainer notre modèle et réaliser une cross-validation pour qualibrer les hyper-paramètres (le paramètre λ de régularisation), et un set de test sur lequel on pourra vérifier les performances de notre modèle. On prendra environ 75% de nos données pour l'entrainement, soit 270 individus. Nous faisons bien attention de prendre ces valeurs aléatoirement dans nos données, pour ne pas avoir ensuite des ensembles de train et test avec une seule valeur de Y, et également que nos ensembles de validation croisée soient représentatifs des données globales.

```
set.seed(1234)
ENS.TRAIN = sample(1:364, 270)
TRAIN = data[ENS.TRAIN,]
TEST = data[-ENS.TRAIN,]
```

Nous pouvons bien vérifier que nous avons créer deux tableaux : TRAIN, une table de 270x8651 valeurs, et TEST, une table de 94x8651 valeurs.

Sur l'échantillon de TRAIN, on va le diviser en 15 sous-ensemble pour la validation croisée. On définit par avance les fonction nécessaires pour la validation croisée :

```
PARTITION = sample(rep(1:15, rep(18,15)), 270)
```

On crée une variable des noms des colonnes :

```
var <- ls(data)[1:8650]
```

On va ensuite utiliser le package glmnet pour réaliser une régression logistique non régularisée pour le moment :

```
glm1 <- glmnet(x = TRAIN[, var] %>% as.matrix, y = TRAIN[,"Y"], lambda=0, family = "binomial")
```

```
print(glm1)
```

```
##
## Call:  glmnet(x = TRAIN[, var] %>% as.matrix, y = TRAIN[, "Y"], family = "binomial",      lambda = 0)
##
##      Df %Dev Lambda
## 1 8650  100      0
```

A l'aide de ce modèle, nous prédisons les paramètres de notre modèle :

```
predict(glm1, type="coef", "lambda.min", allCoef = TRUE)
```

```
glm1p <- predict(glm1, newx = TRAIN[,var] %>% as.matrix, s = "lambda.min")
```

```
glm1p2 = 1*(glm1p>0)
fa_glm1p = factor(glm1p2)
fa_Y_TRAIN = factor(TRAIN$Y)
confusionMatrix(fa_glm1p, fa_Y_TRAIN)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 140   0
##          1   0 130
##
##                Accuracy : 1
##                  95% CI : (0.9864, 1)
##     No Information Rate : 0.5185
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5185
##          Detection Rate : 0.5185
##    Detection Prevalence : 0.5185
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : 0
##
```

Nous obtenons une classification parfaite, mais probablement avec un sur-apprentissge important. En effet, nous avons entrainé un modèle avec 8650 paramètres. Il est donc nécessaire de régulariser à l'aide du paramètre lambda, et de l'estimer à l'aide de la validation croisée.

Nous commençons par rechercher lambda par validation croisée :

```
cv.glmn1 <- cv.glmnet(x= TRAIN[,var] %>% as.matrix, y = TRAIN[,"Y"], alpha = 0, nfolds = 15, foldid = P
```

```
plot(cv.glmn1)
```

```
glmn1.0 <- glmnet(x = TRAIN[, var] %>% as.matrix, y = TRAIN[,"Y"], alpha = 0, family = "binomial")
plot(glmn1.0, xvar = "lambda", label = FALSE, xlab = ~ log(lambda))
abline( v = log(cv.glmn1$lambda.min), col = "red", lty = 2)
```

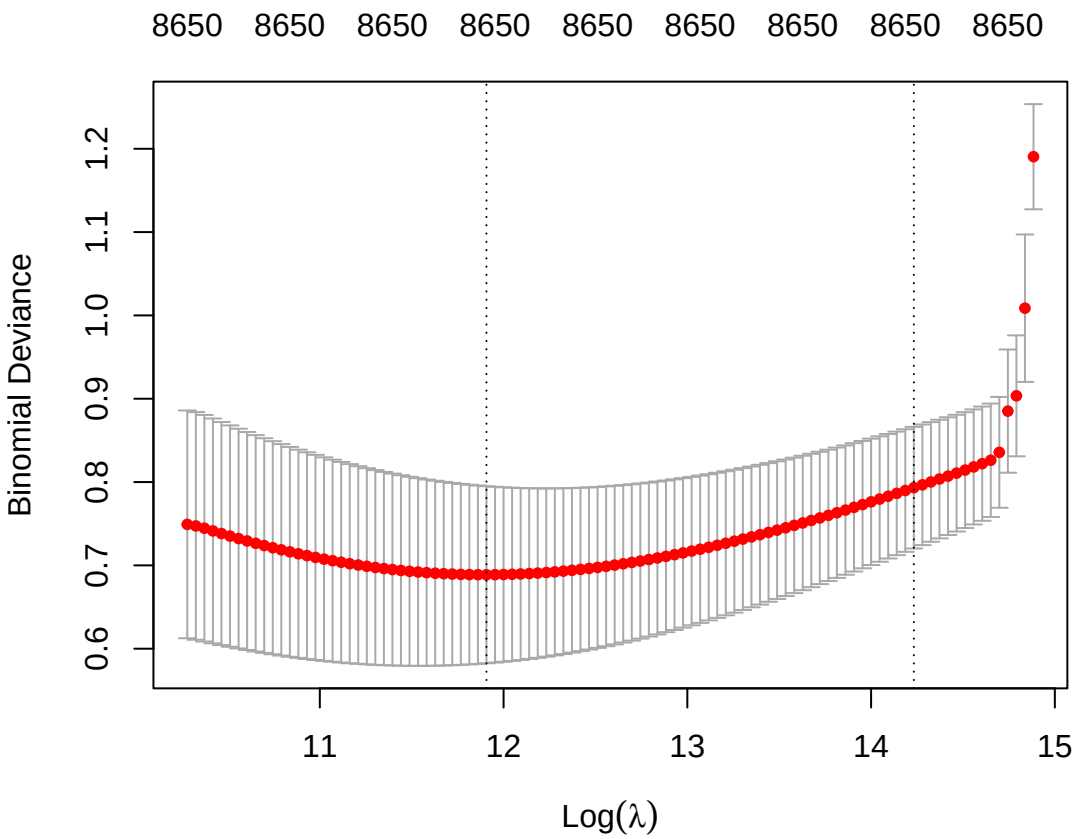

Figure 1: Binomial Deviance funcion of the regularization parameter

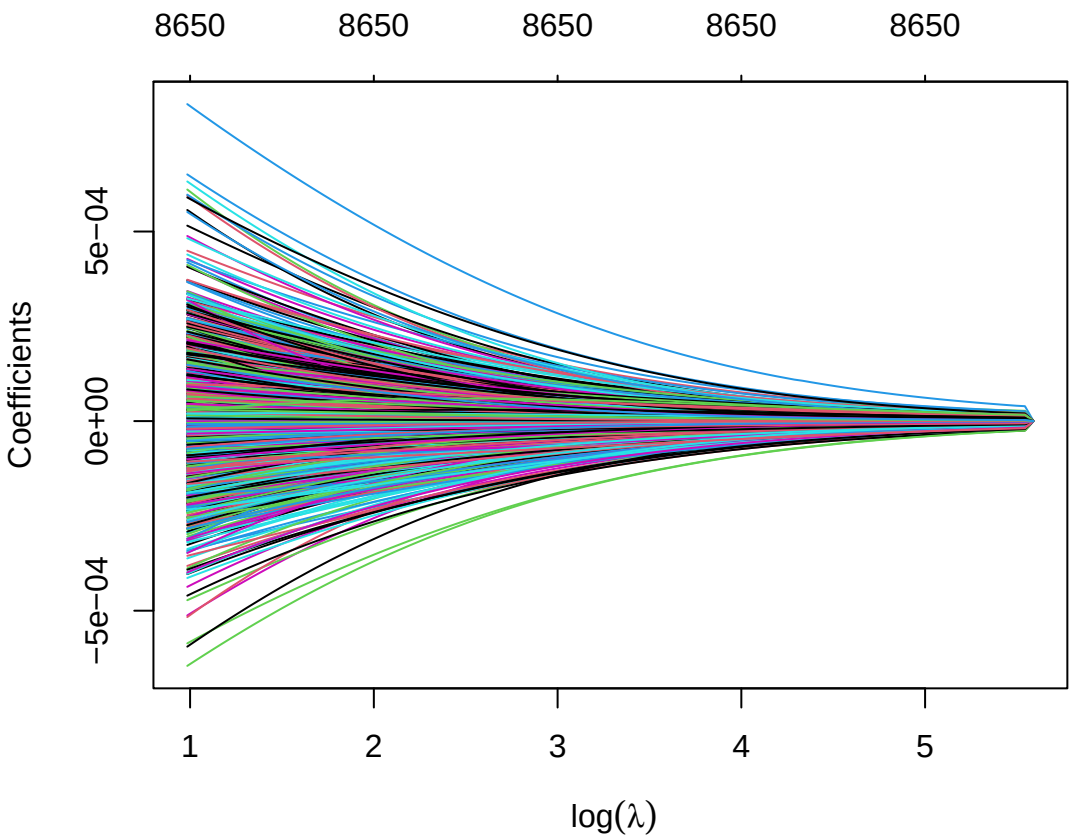

Figure 2: Importance of the coefficients function of the regularization parameter

La validation croisée a été appliquée pour trouver le paramètre lambda. On fait désormais une prédiction basée sur ce modèle.

```
glmn1p <- predict(cv.glmn1, newx = TRAIN[,var] %>% as.matrix, s = "lambda.min")
```

```
glmn1p2 = 1*(glmn1p>0)
fa_glmn1p = factor(glmn1p2)
fa_Y_TRAIN = factor(TRAIN$Y)
confusionMatrix(fa_glmn1p, fa_Y_TRAIN)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 139   1
##          1   1 129
##
##                Accuracy : 0.9926
##                  95% CI : (0.9735, 0.9991)
##     No Information Rate : 0.5185
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9852
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9929
##             Specificity : 0.9923
##          Pos Pred Value : 0.9929
##          Neg Pred Value : 0.9923
##              Prevalence : 0.5185
##          Detection Rate : 0.5148
##    Detection Prevalence : 0.5185
##       Balanced Accuracy : 0.9926
##
##        'Positive' Class : 0
##
```

Nous passons désoramais à la phase de validation du modèle avec l'échantillon test.

```
glm1tp <- predict(cv.glmn1, newx = TEST[,var] %>% as.matrix, s = "lambda.min")
```

```
glm1tp2 = 1*(glm1tp>0)
fa_glmn1tp = factor(glm1tp2)
fa_Y_TEST = factor(TEST$Y)
confusionMatrix(fa_glmn1tp, fa_Y_TEST)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 43  6
```

```
##          1  5 40
##
##                Accuracy : 0.883
##                  95% CI : (0.8003, 0.9401)
##     No Information Rate : 0.5106
##     P-Value [Acc > NIR] : 1.825e-14
##
##                   Kappa : 0.7657
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.8958
##             Specificity : 0.8696
##          Pos Pred Value : 0.8776
##          Neg Pred Value : 0.8889
##              Prevalence : 0.5106
##          Detection Rate : 0.4574
##    Detection Prevalence : 0.5213
##       Balanced Accuracy : 0.8827
##
##        'Positive' Class : 0
##
```

Finalement, avec ce modèle, nous arrivons à avoir une précision de 88%, ce qui semble être un résultat acceptable. Il pourrait cependant être remis en question et retravailler, si les médecins ne veulent pas avoir de cas de faux négatifs. En effet, il semble logique que les médecins veuillent détecter tous les cas d'Alzheimer sans en oublier, quitte à avoir plus de faux-positifs.

4. Cas pratique - Résolution avec Newton-Raphson

Nous allons dans cette partie mettre en place l'algorithme de Newton-Raphson pour nos données, et nous pourrons comparer les performances obtenues par les deux méthodes.

Nous définissons l'algorithme de Newton-Raphson, dans le cas de la régularisation avec un paramètre lambda. Cet algorithme va donc reposer sur l'équation obtenue à la question 8 de la partie 2. L'avantage de cet algorithme est que nous ne devons plus calculer $(X^T V X)^{-1}$, qui est l'inverse d'une matrice de dimension 8650x8650, mais l'inverse de $X^T X$ de dimension 364x364. De plus, cette opération n'a besoin d'être réalisé qu'une seule fois au début de l'algorithme, ce qui permet de gagner beaucoup en performance. Nous pourrons ainsi déterminer α_λ, puis $\beta_\lambda = X^T \alpha_\lambda$, puis $y_{pred} = \pi_\beta(X)$.

```
myLR = function(X, y, lambda, tolerance = 1e-6, max.iter=200){
  X = cbind(1, X)
  alpha_s = rep(0, NROW(X))
  pi = runif(NROW(X), 0, 1)
  Z = solve(X %*% t(X) + 4*lambda*diag(NROW(X)))
  iter = 1
  made.changes = TRUE
  while (made.changes & (iter < max.iter))
  {
    iter = iter + 1
    made.changes <- FALSE
    alpha_s_plus_1 = alpha_s + 4*(y-pi-lambda*alpha_s)%*%Z
    alpha_s_plus_1 = unlist(as.list(alpha_s_plus_1))
```

```
    beta_s_plus_1 = alpha_s_plus_1 %*% X
    pi = drop(1/(1+exp(-beta_s_plus_1 %*% t(X))))
    relative.change = drop(crossprod(alpha_s_plus_1 - alpha_s))/drop(crossprod(alpha_s))
    made.changes = (relative.change > tolerance)
    alpha_s = alpha_s_plus_1
  }
  #print(paste("The Newton-Raphson algorithm converges after",iter, "iterations"))
  return(list(alpha = alpha_s, proba = pi))
}
```

Testons cet algorithme sur notre set d'entrainement :

```
X_train = TRAIN[var]
y_train = TRAIN$Y
X_train = data.matrix(X_train)
y_train = data.matrix(y_train)
y_train = unlist(as.list(t(y_train)))
res = myLR(X = X_train, y = y_train, lambda = 1, max.iter = 200)
y_pred = res$proba
alpha = res$alpha
beta = alpha %*% X_train
```

Pour mesurer la validité de notre modèle, nous définissons la fonction MSE, pour calculer l'erreur quadratique moyenne entre nos prédictions et les vrais labels.

```
MSE = function (y_train, y_pred){
  err = mean((y_train-y_pred)^2)
  return (err)
}
```

```
error1 =MSE(y_train, y_pred)
```

Nous obtenons donc une erreur de 2.5421686×10^{-6}‘ en prenant le MSE comme critère, avec $\lambda = 1$.

Maintenant que notre algorithme de Newton-Raphson est implémenté, nous pouvons mettre en place la validation croisée pour déterminer le meilleur paramètre de régularisation λ.

```
nb_fold = 5 #nombre d'ensembles pour la validation croisée
segment = sample(rep(1:nb_fold, each = NROW(TRAIN)/nb_fold))
```

Après plusieurs tests exploratoires, nous avons remarqué que l'algorithme de Newton-Raphson convergeait autour des 150 itérations. En conservant un nombre d'itérations maximum de 200, on s'assure ainsi de la convergence du modèle pour toutes les valeurs de λ testées. Nous allons commencer par parcourir un large panel de valeur de λ pour trouver le minimum de l'erreur en fonction de λ.

```
lambda = logseq(1, 1e+9, 11)
L = length(lambda)
err = rep(0, L)
MSE_k = rep(0, nb_fold)
for (i in 1:length(lambda)) {
  for (k in 1:nb_fold) {
    xtrain = X_train[-which(segment==k),]
```

```
    ytrain = y_train[-which(segment==k)]
    xtest = X_train[which(segment==k),]
    ytest = y_train[which(segment==k)]
    res = myLR(X = xtrain, y = ytrain, lambda = lambda[i], max.iter = 200)
    alpha = res$alpha
    beta = alpha %*% xtrain
    ytest_pred = drop(1/(1+exp(-beta %*% t(xtest))))
    MSE_k[k] <- MSE(ytest_pred, ytest)
  }
  err[i] <- mean(MSE_k)
  MSE_k = rep(0, nb_fold)
}
```

```
plot(log(lambda), err)
```

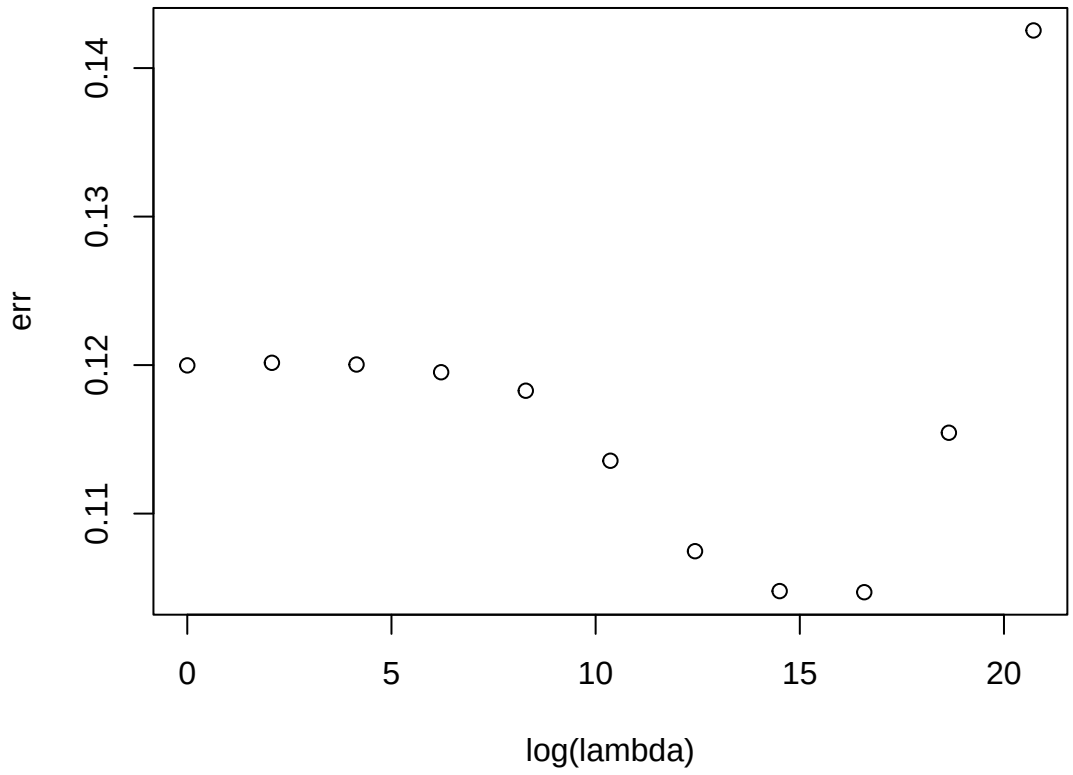

Figure 3: Error function of the parameter of regularization

Nous prendrons donc la valeur de lambda pour laquelle nous avons l'erreur minimale :

```
lambda_hat <- lambda[1]
err_min <- err[1]
```

```
for (i in 1:length(err)){
if (err[i]<err_min){
err_min <- err[i]
lambda_hat <- lambda[i]
}
}
lambda_hat
```

```
## [1] 15848932
```

```
err_min
```

```
## [1] 0.1047063
```

```
logL = log(lambda_hat)
```

Refaisons une analyse de λ plus précise, autour de la valeur trouvée précédemment.

```
lambda = logseq(lambda_hat/10, 5*lambda_hat, 10)
L = length(lambda)
err = rep(0, L)
MSE_k = rep(0, nb_fold)
for (i in 1:length(lambda)) {
  for (k in 1:nb_fold) {
    xtrain = X_train[-which(segment==k),]
    ytrain = y_train[-which(segment==k)]
    xtest = X_train[which(segment==k),]
    ytest = y_train[which(segment==k)]
    res = myLR(X = xtrain, y = ytrain, lambda = lambda[i], max.iter = 200)
    alpha = res$alpha
    beta = alpha %*% xtrain
    ytest_pred = drop(1/(1+exp(-beta %*% t(xtest))))
    MSE_k[k] <- MSE(ytest_pred, ytest)
  }
  err[i] <- mean(MSE_k)
  MSE_k = rep(0, nb_fold)
}
```

```
plot(log(lambda), err)
```

```
lambda_hat <- lambda[1]
err_min <- err[1]
for (i in 1:length(err)){
if (err[i]<err_min){
err_min <- err[i]
lambda_hat <- lambda[i]
}
}
lambda_hat
```

```
## [1] 5838796
```

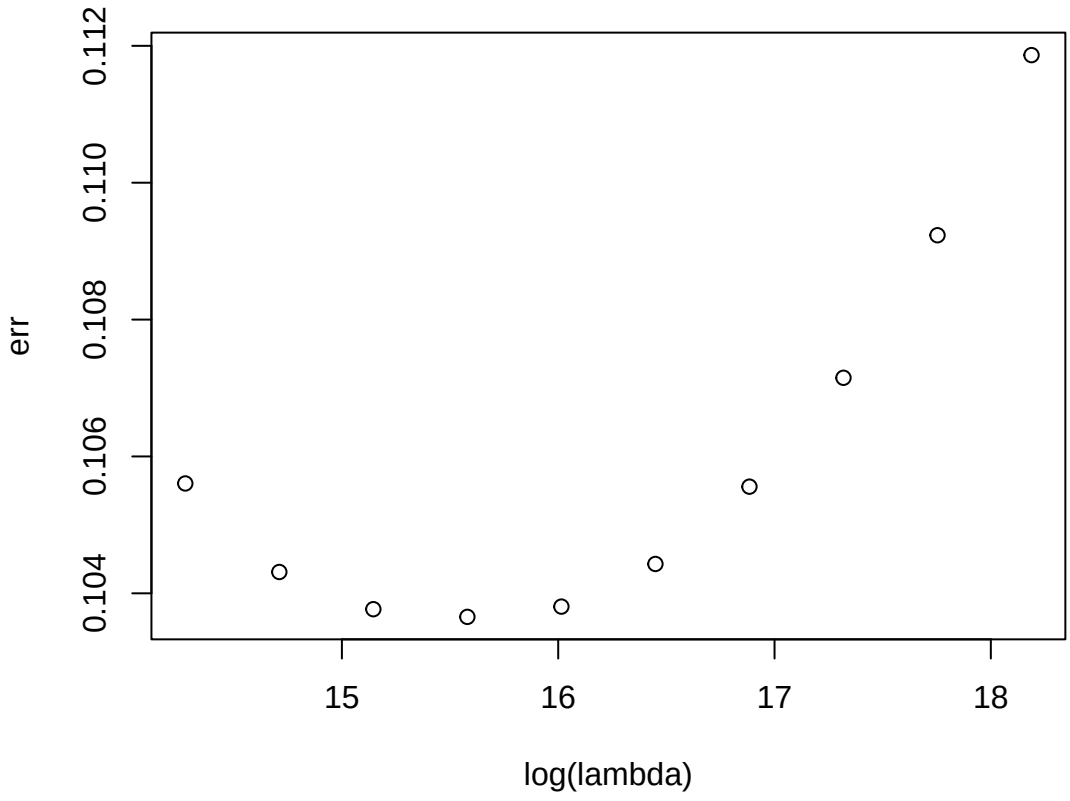

Figure 4: Error function of the parameter of regularization

```
err_min
```

```
## [1] 0.1036558
```

```
logL = log(lambda_hat)
```

Nous avons donc $\hat{\lambda} = 5.8387964 \times 10^6$, ce qui donne $log(\hat{\lambda}) = 15.5800352$, ce qui correspond en ordre de grandeur aux valeurs données par la résolution avec glmnet. Cette valeur de λ semble importante, mais elle paraît cohérente, car nous avons des valeurs de X allant jusqu'à 10^3. Donc les valeurs dans le produit $X^T X$ peuvent atteindre 10^6. Il faut donc avoir des valeurs de λ du même ordre de grandeur pour avoir une régularisation efficace et bien dimensionnée.

Pour finir cette étude, testons notre régression sur le set de test :

```
X_test = TEST[var]
y_test = TEST$Y
X_test = data.matrix(X_test)
y_test = data.matrix(y_test)
y_test = unlist(as.list(t(y_test)))
res = myLR(X = X_test, y = y_test, lambda = lambda_hat, max.iter = 200)
y_pred = res$proba
print(MSE(y_test, y_pred))
```

```
## [1] 0.001896039
```

```
y_p = 1*(y_pred>0.5)
fa_y_p = factor(y_p)
fa_y_test = factor(y_test)
confusionMatrix(fa_y_p, fa_y_test)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 48  0
##          1  0 46
##
##                Accuracy : 1
##                  95% CI : (0.9615, 1)
##     No Information Rate : 0.5106
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5106
##          Detection Rate : 0.5106
```

```
##    Detection Prevalence : 0.5106
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : 0
##
```

```
fit.roc = roc(y_test, y_p)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
plot(fit.roc, print.thres = "best")
```

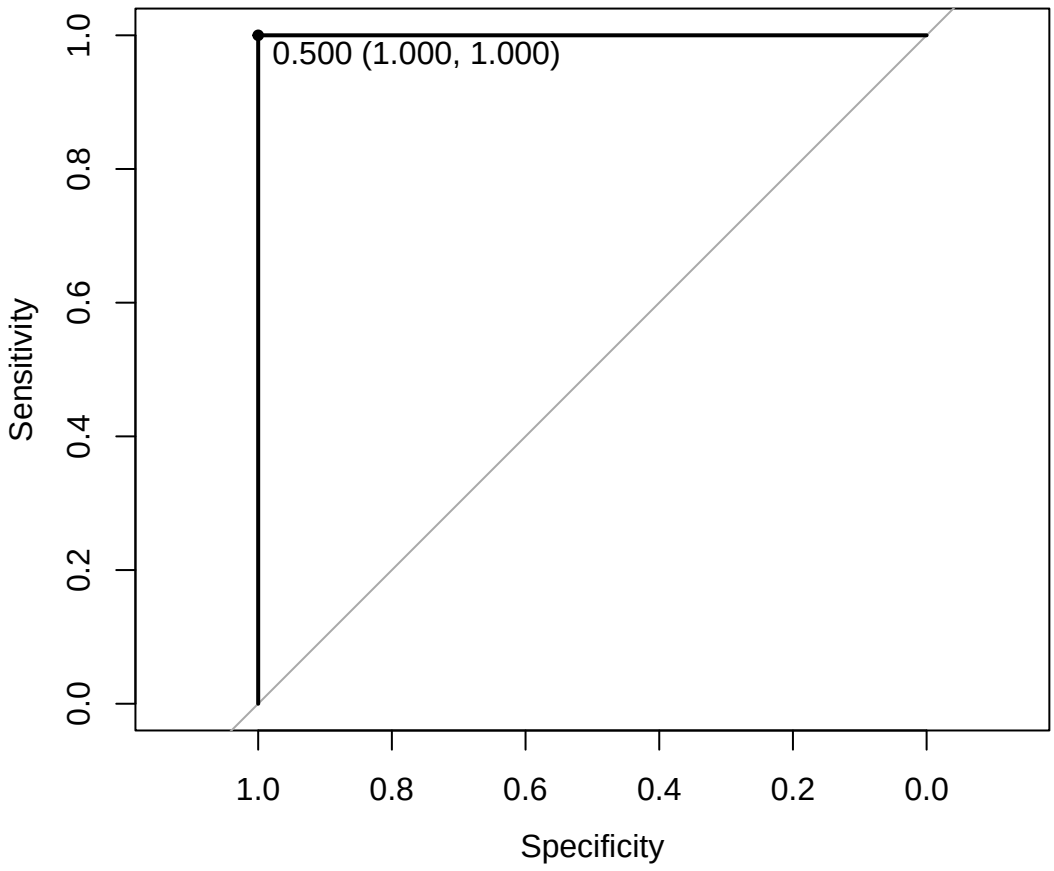

Figure 5: Confusion Matrix and Metrics

```
auc(fit.roc)
```

```
## Area under the curve: 1
```

Nos résultats semblent désormais extrêmement convaincant pour prédire le statut du patient.

5. Conclusion

Dans cette étude, nous avons donc pu mettre en application l'aglorithme de Newton-Raphson, et comparer ses résultats aux fonctions déjà définies sur R comme glmnet. Nous avons pu voir que nous obtenions des valeurs dans les mêmes ordres de grandeur, et de très bons résultats de classification. Nous avons surtout pu voir l'importance de travailler dans l'algorithme avec les α plutôt que les β, car nous avons des calculs beaucoup moins complexes à réaliser, et certains peuvent même n'être réalisés qu'une seule fois. Nous gagnons donc énormément en performance.