Regression\_Logistique

# Le modèle de régression logistique

La régression logistique est un modèle prédictif dont le but est de prédire/expliquer les valeurs prises par une variable cible qualitative. Dans notre cas, la variable cible y est de type binaire (yes/no) et désigne si le client à souscrit ou non un dépôt à terme. On parle donc de régression logistique binaire.

Il est possible de pénaliser le modèle de régression logistique. En effet, il y a deux types de régression pénalisée:  
- la régression Ridge: pénalise la magnitude absolue des coefficients  
- la régression Lasso: pénalise le nombre de coefficient différents de 0  
Ces paramètres sont utilisés pour ajuster les coefficients de la régression à l’aide de plusieurs paramètres comme alpha qui permet d’ajuster le modèle Ridge ou Lasso et lambda qui permet de contrôler la pénalité.

Dans un premier temps, nous allons charger et préparer les données. Nous essaierons ensuite plusieurs modèles de régression logistique, notamment des modèles pénalisés. Enfin, nous comparrons les performances de nos modèles.

# Chargement des packages

library(dplyr)  
library(glmnet)  
library(caTools)  
library(csv)

# Préparation des données

set.seed(123)  
#Chargement des données  
#data<-read.csv2("C:/Users/Axelle/Desktop/M/03\_SISE/05\_MACHINE LEARNING/Projet/bank.csv")  
data<-read.csv2("bank.csv")  
  
#Recodage de la variable cible  
data$y = recode(data$y, "no" = 0, "yes" = 1)  
prop.table(table(data$y))

##   
## 0 1   
## 0.88476 0.11524

Nous avons recodé la variable cible en variable binaire 0/1 ou 0 désigne la classe “no” et 1 désigne la classe “yes”.  
88.47% de nos observations sont de la classe 0. Le modèle par défaut consisterait donc à prédire systématiquement la classe majoritaire 0. Donc, avec la prédiction par défaut, le taux d’erreur serait de 11.52%.

# Echantillonage

Nous séparons les données en un échantillon d’apprentissage et un échantillon de test puis nous séparons la variable cible et les variables explicatives dans des variables différentes.

#Echantillonage des données  
ech = sort(sample(nrow(data), nrow(data)\*.7))  
train = data[ech, ]  
test = data[-ech, ]  
  
#Definition variables explicatives/variable cible  
Xtrain = train[, -17]  
ytrain = train[, 17]  
Xtest = test[, -17]  
ytest = test[, 17]

# La régression logistique

Nous essayons d’abord de prédire nos données avec un modèle de régression logistique sans pénalisation (avec lambda=0).

#Modèle de régression logistique  
reg = glmnet(model.matrix(~., Xtrain), ytrain, family="binomial", lambda=0)   
#Prédiction sur les données test  
pred = predict(reg, newx = model.matrix(~., Xtest), type="class", s=c(0))  
#Matrice de confusion  
cm = table(pred, ytest); cm

## ytest  
## pred 0 1  
## 0 1170 98  
## 1 34 55

#Taux d'erreur  
err = (cm[1,2] + cm[2,1])/sum(cm); err

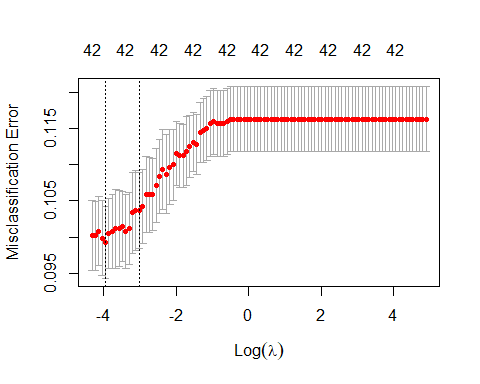
## [1] 0.0972734

Le taux d’erreur est de 9.73% qui est donc meilleur que la prédiction par défaut.

# La régression Ridge

La régression ridge correspond à lambda>0 et alpha=0.  
Nous allons déterminer la valeur optimal de lambda par cross validation en 10 folds.

# Optimisation du paramètre lambda  
cv.ridge = cv.glmnet(model.matrix(~., Xtrain), ytrain, family="binomial", type.measure="class", nfolds=10, alpha=0, keep=TRUE)  
plot(cv.ridge)



Le graphique met en relation les valeurs de log(λ) avec le taux d’erreur moyen en validation croisée.  
Le premier trait en pointillé représente le lambda qui minimise l’erreur. Il vaut -3.94.  
Le deuxième trait en pointillé représente la plus grande valeur de lambda pour laquelle l’erreur moyenne en validation croisée est inférieure à la borne haute de l’intervalle de confiance de l’erreur optimale. Il vaut -3.01.

lambda\_min = cv.ridge$lambda.min  
lambda\_1se = cv.ridge$lambda.1se

Nous allons essayer de faire les prédictions avec ces deux valeurs de lambda.

#Prédiction  
pred\_ridge = predict(cv.ridge , newx = model.matrix(~., Xtest), s=c(lambda\_min, lambda\_1se), type="class")  
#Matrices de confusion  
cm\_ridge1 = table(pred\_ridge[,1], ytest)  
cm\_ridge2 = table(pred\_ridge[,2], ytest)  
#Taux d'erreur  
err\_ridge1 = (cm\_ridge1[1,2] + cm\_ridge1[2,1])/sum(cm\_ridge1); err\_ridge1

## [1] 0.09801032

err\_ridge2 = (cm\_ridge2[1,2] + cm\_ridge2[2,1])/sum(cm\_ridge2); err\_ridge2

## [1] 0.1024318

Les deux modèles Ridge font mieux que la prédiction par défaut mais sont moins bien que le modele precedent (sans pénalisation).  
Si on compare les deux modèles de Ridge, le premier est un peu meilleur. Le modèle le plus pénalisé n’est donc pas le meilleur choix dans notre cas.

# La régression elasticnet

La régression ridge correspond à lambda>0 et alpha>0.  
Nous allons déterminer la valeur optimal de lambda et alpha.  
Nous commençons par cross valider notre modèle.

#Modèles  
for (i in 1:10) {  
 assign(paste("fit", i, sep=""), cv.glmnet(model.matrix(~., Xtrain), ytrain, type.measure="class", alpha=i/10,family="binomial"))  
}

Prédiction avec les différents modèles

#Predictions  
pred\_enet1 = predict(fit1, s=c(fit1$lambda.1se, fit1$lambda.min), newx=model.matrix(~., Xtest), type="class")  
pred\_enet2 = predict(fit2, s=c(fit2$lambda.1se, fit2$lambda.min), newx=model.matrix(~., Xtest), type="class")  
pred\_enet3 = predict(fit3, s=c(fit3$lambda.1se, fit3$lambda.min), newx=model.matrix(~., Xtest), type="class")  
pred\_enet4 = predict(fit4, s=c(fit4$lambda.1se, fit4$lambda.min), newx=model.matrix(~., Xtest), type="class")  
pred\_enet5 = predict(fit5, s=c(fit5$lambda.1se, fit5$lambda.min), newx=model.matrix(~., Xtest), type="class")  
pred\_enet6 = predict(fit6, s=c(fit6$lambda.1se, fit6$lambda.min), newx=model.matrix(~., Xtest), type="class")  
pred\_enet7 = predict(fit7, s=c(fit7$lambda.1se, fit7$lambda.min), newx=model.matrix(~., Xtest), type="class")  
pred\_enet8 = predict(fit8, s=c(fit8$lambda.1se, fit8$lambda.min), newx=model.matrix(~., Xtest), type="class")  
pred\_enet9 = predict(fit9, s=c(fit9$lambda.1se, fit9$lambda.min), newx=model.matrix(~., Xtest), type="class")  
pred\_enet10 = predict(fit10, s=c(fit10$lambda.1se, fit10$lambda.min), newx=model.matrix(~., Xtest), type="class")

Calcul des matrices de confusion

#Matrices de confusion  
cm\_enet1\_1se = table(pred\_enet1[,1], ytest)  
cm\_enet2\_1se = table(pred\_enet2[,1], ytest)  
cm\_enet3\_1se = table(pred\_enet3[,1], ytest)  
cm\_enet4\_1se = table(pred\_enet4[,1], ytest)  
cm\_enet5\_1se = table(pred\_enet5[,1], ytest)  
cm\_enet6\_1se = table(pred\_enet6[,1], ytest)  
cm\_enet7\_1se = table(pred\_enet7[,1], ytest)  
cm\_enet8\_1se = table(pred\_enet8[,1], ytest)  
cm\_enet9\_1se = table(pred\_enet9[,1], ytest)  
cm\_enet10\_1se = table(pred\_enet10[,1], ytest)  
  
cm\_enet1\_min = table(pred\_enet1[,2], ytest)  
cm\_enet2\_min = table(pred\_enet2[,2], ytest)  
cm\_enet3\_min = table(pred\_enet3[,2], ytest)  
cm\_enet4\_min = table(pred\_enet4[,2], ytest)  
cm\_enet5\_min = table(pred\_enet5[,2], ytest)  
cm\_enet6\_min = table(pred\_enet6[,2], ytest)  
cm\_enet7\_min = table(pred\_enet7[,2], ytest)  
cm\_enet8\_min = table(pred\_enet8[,2], ytest)  
cm\_enet9\_min = table(pred\_enet9[,2], ytest)  
cm\_enet10\_min = table(pred\_enet10[,2], ytest)

Calcul des taux d’erreur

#Taux d'erreur  
err\_enet1\_1se = (cm\_enet1\_1se[1,2] + cm\_enet1\_1se[2,1])/sum(cm\_enet1\_1se)  
err\_enet2\_1se = (cm\_enet2\_1se[1,2] + cm\_enet2\_1se[2,1])/sum(cm\_enet2\_1se)  
err\_enet3\_1se = (cm\_enet3\_1se[1,2] + cm\_enet3\_1se[2,1])/sum(cm\_enet3\_1se)  
err\_enet4\_1se = (cm\_enet4\_1se[1,2] + cm\_enet4\_1se[2,1])/sum(cm\_enet4\_1se)  
err\_enet5\_1se = (cm\_enet5\_1se[1,2] + cm\_enet5\_1se[2,1])/sum(cm\_enet5\_1se)  
err\_enet6\_1se = (cm\_enet6\_1se[1,2] + cm\_enet6\_1se[2,1])/sum(cm\_enet6\_1se)  
err\_enet7\_1se = (cm\_enet7\_1se[1,2] + cm\_enet7\_1se[2,1])/sum(cm\_enet7\_1se)  
err\_enet8\_1se = (cm\_enet8\_1se[1,2] + cm\_enet8\_1se[2,1])/sum(cm\_enet8\_1se)  
err\_enet9\_1se = (cm\_enet9\_1se[1,2] + cm\_enet9\_1se[2,1])/sum(cm\_enet9\_1se)  
err\_enet10\_1se = (cm\_enet10\_1se[1,2] + cm\_enet10\_1se[2,1])/sum(cm\_enet10\_1se)  
err\_enet\_1se = c(err\_enet1\_1se, err\_enet2\_1se, err\_enet3\_1se, err\_enet4\_1se, err\_enet5\_1se, err\_enet6\_1se, err\_enet7\_1se, err\_enet8\_1se, err\_enet9\_1se, err\_enet10\_1se)  
  
err\_enet1\_min = (cm\_enet1\_min[1,2] + cm\_enet1\_min[2,1])/sum(cm\_enet1\_min)  
err\_enet2\_min = (cm\_enet2\_min[1,2] + cm\_enet2\_min[2,1])/sum(cm\_enet2\_min)  
err\_enet3\_min = (cm\_enet3\_min[1,2] + cm\_enet3\_min[2,1])/sum(cm\_enet3\_min)  
err\_enet4\_min = (cm\_enet4\_min[1,2] + cm\_enet4\_min[2,1])/sum(cm\_enet4\_min)  
err\_enet5\_min = (cm\_enet5\_min[1,2] + cm\_enet5\_min[2,1])/sum(cm\_enet5\_min)  
err\_enet6\_min = (cm\_enet6\_min[1,2] + cm\_enet6\_min[2,1])/sum(cm\_enet6\_min)  
err\_enet7\_min = (cm\_enet7\_min[1,2] + cm\_enet7\_min[2,1])/sum(cm\_enet7\_min)  
err\_enet8\_min = (cm\_enet8\_min[1,2] + cm\_enet8\_min[2,1])/sum(cm\_enet8\_min)  
err\_enet9\_min = (cm\_enet9\_min[1,2] + cm\_enet9\_min[2,1])/sum(cm\_enet9\_min)  
err\_enet10\_min = (cm\_enet10\_min[1,2] + cm\_enet10\_min[2,1])/sum(cm\_enet10\_min)  
err\_enet\_min = c(err\_enet1\_min, err\_enet2\_min, err\_enet3\_min, err\_enet4\_min, err\_enet5\_min, err\_enet6\_min, err\_enet7\_min, err\_enet8\_min, err\_enet9\_min, err\_enet10\_min)

Comparaion des taux d’erreur

df\_err\_enet = data.frame(err\_enet\_1se,err\_enet\_min)  
df\_err\_enet

## err\_enet\_1se err\_enet\_min  
## 1 0.1061164 0.09801032  
## 2 0.1009580 0.09579956  
## 3 0.1039057 0.09801032  
## 4 0.1016949 0.09653648  
## 5 0.1031688 0.09579956  
## 6 0.1009580 0.09727340  
## 7 0.1016949 0.09727340  
## 8 0.1002211 0.09506264  
## 9 0.1024318 0.09579956  
## 10 0.1024318 0.09653648

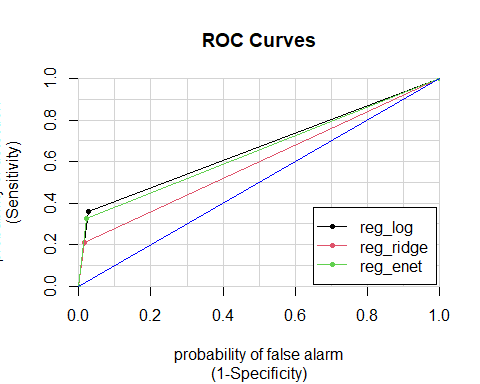
Nous remarquons que les modèles les plus pénalisés (ceux avec lambda.1se) ont un taux d’erreur plus elevé et sont donc moins bon. Le meilleur de nos modèles a un taux d’erreur de 0.0950626.

Récupérons à présents nos 3 meilleurs modèles. Nous allons afficher la courbe ROC afin de comparer ces modèles.

all\_pred = data.frame(as.numeric(pred), as.numeric(pred\_ridge[,2]), as.numeric(pred\_enet8[,2]))  
colnames(all\_pred) = c("reg\_log", "reg\_ridge", "reg\_enet")  
colAUC(all\_pred, ytest, plotROC = TRUE)

## reg\_log reg\_ridge reg\_enet  
## 0 vs. 1 0.665619 0.5971001 0.6526014

abline(0,1, col = "blue")



Nous remarquons que nos trois modèles font mieux que l’aléatoire. Le meilleur des trois modèles est le modèle non pénalisé suivi du modèle elasticnet puis du modèle Ridge.  
Les aires sous la courbes sont compris entre 0.6 et 0.67. Il ne s’agit donc pas d’exellents modèles.