Modèles Linéaires

TD 7 : Modèles généralisés : Régression logistique
Licence 3 MIASHS

Guillaume Metzler & Ugo Zennaro
Institut de Communication (ICOM)
Université de Lyon, Université Lumière Lyon 2
Laboratoire ERIC UR 3083, Lyon, France
guillaume.metzler@univ-lyon2.fr ; ugo.zennaro@univ-lyon2.fr

Résumé

Nous commençons notre étude des modèles linéaires généralisés avec le modèle de régression logistique. Les détails de la présentation de ce modèle ont été donnés en cours et nous ne reviendrons pas sur la façon dont sont estimés les paramètres du modèle.
Plus précisément, nous allons

- Apprendre à effectuer une régression logistique sous R
- Analyser les résultats de la régression logistique
- Construire un bon modèle de régression logistique
- Evaluer ses performances et sa capacité à généraliser sur de nouvelles données.

1 Généralités sur la régression logistique

On rappelle que notre modèle de régression logistique repose sur l'estimation du *logit* par une relation linéaire des covariables, *i.e.* on modélise le logarithme de l'*odd ration* par un modèle linéaire

$$\ln\left(\frac{Pr[Y=1 \mid X=\mathbf{x}]}{Pr[Y=0 \mid X=\mathbf{x}]}\right) = \beta_0 + \beta_1 X1 + \beta_2 X_2 + \ldots + \beta_p X_p \tag{1}$$

Dans ce type modèle, on ne cherche donc plus à prédire un nombre réel, mais plutôt la probabilité d'appartenance à une classe donnée, dite de *référence*, ici la classe 1.
Le modèle de régression logistique est spécifiquement dédiée à des problèmes de **classification binaire**, où l'on cherche **à ranger les individus dans deux groupes distincts**. Ces groupes sont généralement notés 0 et 1, parfois -1 et 1.

En cours, nous avons vu que ce modèle, contrairement au modèle de régression linéaire classique n'admet de pas de solution explicite et nous devons donc avoir recours à des algorithmes d'optimisation pour approcher la solution.

1. Rappeler ce qu'est le modèle *logit*
   La fonction *logit* est donnée par l'expression suivante :

$$\begin{aligned} \text{logit} : [0,1] &\to \mathbb{R} \\ p &\to \ln\left(\frac{p}{1-p}\right) \end{aligned}$$

   elle modélise une probabilité dans l'espace des réels. La fonction inverse est la fonction logistique qui exprime un réel dans un espace de probabilité :

$$\begin{aligned} \text{logistique} : \mathbb{R} &\to [0,1] \\ Y &\to \frac{1}{1+\exp(-Y)}. \end{aligned}$$

   Le modèle associé vise à estimer Y, l'image de la fonction logit, par un modèle linéaire, et en déduire la proba associée avec la fonction logistique.
2. Montrer, à partir de l'Equation (1), que la probabilité d'appartenir à la classe 1 est égale à

$$Pr(Y = 1 \mid X = \mathbf{x}) = \frac{1}{1+\exp(-(\beta_0 + \beta_1 X1 + \beta_2 X_2 + \ldots + \beta_p X_p))}.$$

   Dans le cas d'une classification binaire on a $Pr(Y = 1) = 1 - Pr(Y = 0)$, ainsi :

$$\begin{aligned} &\ln\left(\frac{Pr[Y=1 \mid X=\mathbf{x}]}{Pr[Y=0 \mid X=\mathbf{x}]}\right) = \beta_0 + \beta_1 X1 + \beta_2 X_2 + \ldots + \beta_p X_p \\ \iff &\ln\left(\frac{Pr[Y=0 \mid X=\mathbf{x}]}{Pr[Y=1 \mid X=\mathbf{x}]}\right) = -(\beta_0 + \beta_1 X1 + \beta_2 X_2 + \ldots + \beta_p X_p) \\ \iff &\frac{1 - Pr[Y=1 \mid X=\mathbf{x}]}{Pr[Y=1 \mid X=\mathbf{x}]} = \exp\left(-(\beta_0 + \beta_1 X1 + \beta_2 X_2 + \ldots + \beta_p X_p)\right) \\ \iff &\frac{1}{Pr[Y=1 \mid X=\mathbf{x}]} - 1 = \exp\left(-(\beta_0 + \beta_1 X1 + \beta_2 X_2 + \ldots + \beta_p X_p)\right) \\ \iff &Pr[Y=1 \mid X=\mathbf{x}] = \frac{1}{1+\exp\left(-(\beta_0 + \beta_1 X1 + \beta_2 X_2 + \ldots + \beta_p X_p)\right)} \end{aligned}$$

3. Rappeler l'expression de la vraisemblance.
   L'expression de la vraisemblance associée est :

$$\mathcal{L}(\boldsymbol{\beta}, S) = \prod_{i=1}^{m} Pr(Y = y_i \mid X = \mathbf{x}_i),$$

$$\downarrow \text{ on sépare } y_i = 0 \text{ et } y_i = 1$$

$$= \prod_{i=1, y_i=1}^{m} Pr(Y = y_i \mid X = \mathbf{x}_i) \times \prod_{i=1, y_i=0}^{m} Pr(Y = y_i \mid X = \mathbf{x}_i),$$

$$\downarrow \text{ on utilise le fait que l'on suit une loi de Bernoulli}$$

$$\downarrow \text{ nous n'avons que deux issues possibles}$$

$$= \prod_{i=1}^{m} \left(\frac{1}{1 + \exp(-\mathbf{x}_i \boldsymbol{\beta})} \right)^{y_i} \times \left(\frac{1}{1 + \exp(\mathbf{x}_i \boldsymbol{\beta})} \right)^{(1-y_i)}.$$

Dans la suite de ce TP, on va travailler sur le jeu de données *glass.csv* que vous pourrez télécharger sur la page du cours.

```
# Renommer le jeu de données
data = read.csv("../data/glass.csv", header = TRUE)
head(data)
```

```
##        X1    X2   X3   X4    X5   X6   X7 X8   X9 y
## 1 1.52101 13.64 4.49 1.10 71.78 0.06 8.75  0 0.00 1
## 2 1.51761 13.89 3.60 1.36 72.73 0.48 7.83  0 0.00 1
## 3 1.51618 13.53 3.55 1.54 72.99 0.39 7.78  0 0.00 1
## 4 1.51766 13.21 3.69 1.29 72.61 0.57 8.22  0 0.00 1
## 5 1.51742 13.27 3.62 1.24 73.08 0.55 8.07  0 0.00 1
## 6 1.51596 12.79 3.61 1.62 72.97 0.64 8.07  0 0.26 1
```

4. Évaluer la proportion d'individus dans chaque classe.

```
table(data$y)/nrow(data)
```

```
##
##         0         1
## 0.6728972 0.3271028
```

2 Un premier modèle de régression

On cherche maintenant à construire un modèle de régression logistique. Sur R nous pouvons apprendre un modèle de régression logistique à l'aide de la commande suivante

```
mymodel = glm(y ~., data = data, family = binomial())
```

On commence par regarder quelques éléments de bases de la régression à l'aide de la fonction :

5. Commenter les sorties du modèle de régression en identifiant les variables significatives.

```
summary(mymodel)
```

```
##
## Call:
## glm(formula = y ~ ., family = binomial(), data = data)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.19096  -0.72806  -0.04975   0.75017   1.82235
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -575.3570   341.4893  -1.685  0.09202 .
## X1            67.1196   192.6359   0.348  0.72752
## X2             3.5201     2.0937   1.681  0.09271 .
## X3             6.1380     2.2075   2.780  0.00543 **
## X4             1.4019     2.3365   0.600  0.54850
## X5             5.0003     2.0758   2.409  0.01600 *
## X6             4.4712     2.4649   1.814  0.06969 .
## X7             4.3862     2.1879   2.005  0.04499 *
## X8             4.9974     2.5373   1.970  0.04889 *
## X9            -0.9779     2.0595  -0.475  0.63489
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 270.54  on 213  degrees of freedom
## Residual deviance: 178.67  on 204  degrees of freedom
```

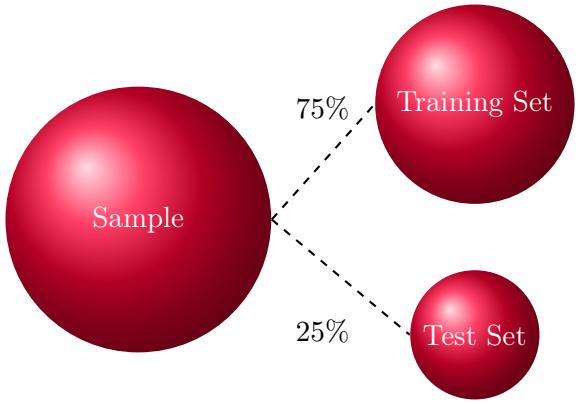


FIGURE 1 – Seperating the dataset into train/test with proportion 75/25 %

```
## AIC: 198.67
##
## Number of Fisher Scoring iterations: 7
```

En prenant $\alpha = 0.05$ comme seuil de rejet de l'hypothèse H_0 :"Le coefficient associé à la i^e variable est nul", les variables significatives sont X2, X3, X4, X5, X6, X7 et X8.

6. Quel critère peut-on utiliser pour évaluer la qualité du modèle et quelle est sa valeur ?

   L'AIC peut être utilisé pour évaluer la qualité du modèle, ici il vaut 237.21.

On va maintenant chercher à mettre en place une procédure d'apprentissage en utilisant le fait que nous devons diviser notre jeu de données en deux ensembles : un ensemble d'*entraînement* et un ensemble *test*, comme illustré en Figure 1.

Nous allons donc apprendre les paramètres du modèle sur l'ensemble d'entraînement et évaluer ses performances sur les données tests (non utilisées pour apprendre).

7. Regarder la fonctionnement de la fonction *CreateDataParitition* du package **caret** de R pour séparer votre jeu de donnés avec les proportions indiquées en Figure 1. Créer votre jeu de données *train* et *test*.

```
set.seed(42)
index_train <- createDataPartition(y=data$y, p = 0.75, list=FALSE)
```

```
library(caret)

## Loading required package: lattice
## Loading required package: ggplot2
## Warning in system("timedatectl", intern = TRUE): running command
'timedatectl' had status 1
```

```
set.seed(42)
index_train <- createDataPartition(y=data$y, p = 0.75, list=FALSE)
train <- data[index_train,]
test <- data[-index_train,]
```

8. Apprendre le modèle sur le jeu de données *train*.

```
mymodel <- glm(y~.,data=train, family = binomial())
```

Pour évaluer les performances du modèles, nous pouvons chercher à évaluer son accuracy, *Acc*.

$$Acc = \frac{TP + TN}{FP + FN + TP + TN}.$$

1. On peut effectuer les prédictions sur les nouvelles données à l'aide de la fonction *predict*

```
proba_test = predict(mymodel,newdata=test, type="response")
summary(proba_test)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## 0.0000296 0.0989279 0.3464575 0.3554105 0.6117580 0.8713435
```

Les valeurs retournées seront des probabilités. Comment attribuer une classe à l'aide ces probabilités ?

On attribuera la classe la plus probable, sachant qu'ici la fonction PRED renvoie la probabilité de $Y = 1$, et qu'on est dans un cas de classification binaire. Ainsi on attribue la classe $Y = 1$ quand la prédiction est superieure à 0.5 et $Y = 0$ autrement.

```
pred <- proba_test > 0.5
table(pred>0.5)/length(pred)
```

```
##
##     FALSE      TRUE
## 0.6981132 0.3018868
```

2. Déterminer l'accuracy sur le jeu de données d'entraînement et sur le jeu de données tests.

```
print(sum(pred == test$y) / length(test$y))
```

```
## [1] 0.7169811
```

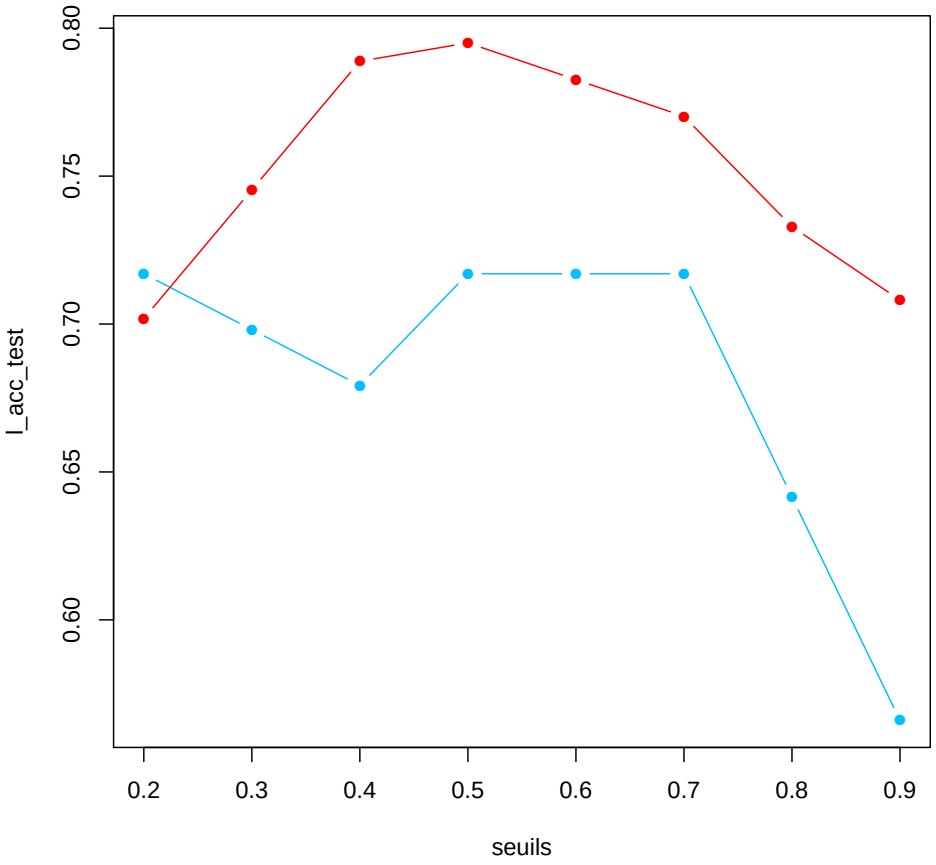
```
proba_train <- predict(mymodel,newdata=train, type="response")
print(sum((proba_train>0.5)== train$y)/ length(train$y))
```

```
## [1] 0.7950311
```

3. Les résultats, en terme d'accuracy, sont-ils stables lorsque l'on bouge le seul de décision ? Faite l'expérience.

```
seuils <- c(0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9)
d_acc <- c()
l_acc_train <- c()
l_acc_test <- c()
for (s in seuils){
  p_train <- proba_train > s
  acc_train <- sum(p_train== train$y)/ length(train$y)
  p_test <- proba_test > s
  acc_test <- sum(p_test== test$y)/ length(test$y)
  l_acc_test <- c(l_acc_test, acc_test)
  l_acc_train <- c(l_acc_train, acc_train)
}

all_pred <- c(l_acc_test, l_acc_train)
plot(seuils, l_acc_test, type = "b", pch = 16, col ="deepskyblue",
     ylim = c(min(all_pred),max(all_pred)))
lines(seuils, l_acc_train,type = "b", pch = 16, col = "red")
```



3 Un modèle de régression simplifié

Reprenez les questions précédentes, en supprimant les variables non significatives et comparer les deux modèles.

```
    step(mymodel)

## Start:  AIC=149.49
## y ~ X1 + X2 + X3 + X4 + X5 + X6 + X7 + X8 + X9
##
##        Df Deviance    AIC
```

```
## - X9    1   129.72 147.72
## - X1    1   129.82 147.82
## - X4    1   130.38 148.38
## <none>      129.49 149.49
## - X6    1   132.67 150.67
## - X2    1   132.83 150.83
## - X7    1   133.47 151.47
## - X8    1   133.63 151.63
## - X5    1   138.11 156.11
## - X3    1   139.05 157.05
##
## Step:  AIC=147.72
## y ~ X1 + X2 + X3 + X4 + X5 + X6 + X7 + X8
##
##        Df Deviance    AIC
## - X1    1   130.13 146.13
## - X4    1   130.94 146.94
## <none>      129.72 147.72
## - X6    1   133.21 149.21
## - X2    1   133.63 149.63
## - X8    1   134.00 150.00
## - X7    1   134.05 150.05
## - X5    1   139.08 155.08
## - X3    1   139.75 155.75
##
## Step:  AIC=146.13
## y ~ X2 + X3 + X4 + X5 + X6 + X7 + X8
##
##        Df Deviance    AIC
## - X4    1   131.71 145.71
## <none>      130.13 146.13
## - X6    1   134.32 148.32
## - X2    1   135.23 149.23
## - X8    1   135.28 149.28
## - X7    1   137.76 151.76
## - X5    1   140.33 154.33
## - X3    1   143.53 157.53
##
## Step:  AIC=145.71
## y ~ X2 + X3 + X5 + X6 + X7 + X8
##
##        Df Deviance    AIC
## <none>      131.71 145.71
```

```
## - X6    1   134.34 146.34
## - X8    1   135.32 147.32
## - X2    1   136.76 148.76
## - X7    1   146.45 158.45
## - X5    1   148.41 160.41
## - X3    1   165.25 177.25
##
## Call:  glm(formula = y ~ X2 + X3 + X5 + X6 + X7 + X8, family = binomial(),
##     data = train)
##
## Coefficients:
## (Intercept)           X2           X3           X5           X6           X7
##    -380.862        2.300        5.071        4.177        3.020        3.156
##          X8
##       4.485
##
## Degrees of Freedom: 160 Total (i.e. Null);  154 Residual
## Null Deviance:     194.4
## Residual Deviance: 131.7  AIC: 145.7
```

```
    model2 <- glm(formula = y ~ X2 + X3 + X5 + X6 + X7 + X8,
        family = binomial(),
        data = train)
    proba_test <- predict(model2,newdata=test, type="response")
    summary(proba_test)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## 0.0000209 0.1044381 0.3260571 0.3546985 0.5483312 0.8431615
```

```
    pred <- proba_test > 0.5
    table(pred>0.5)/length(pred)
```

```
##
##     FALSE      TRUE
## 0.6981132 0.3018868
```

```
    print(sum(pred == test$y) / length(test$y))
```

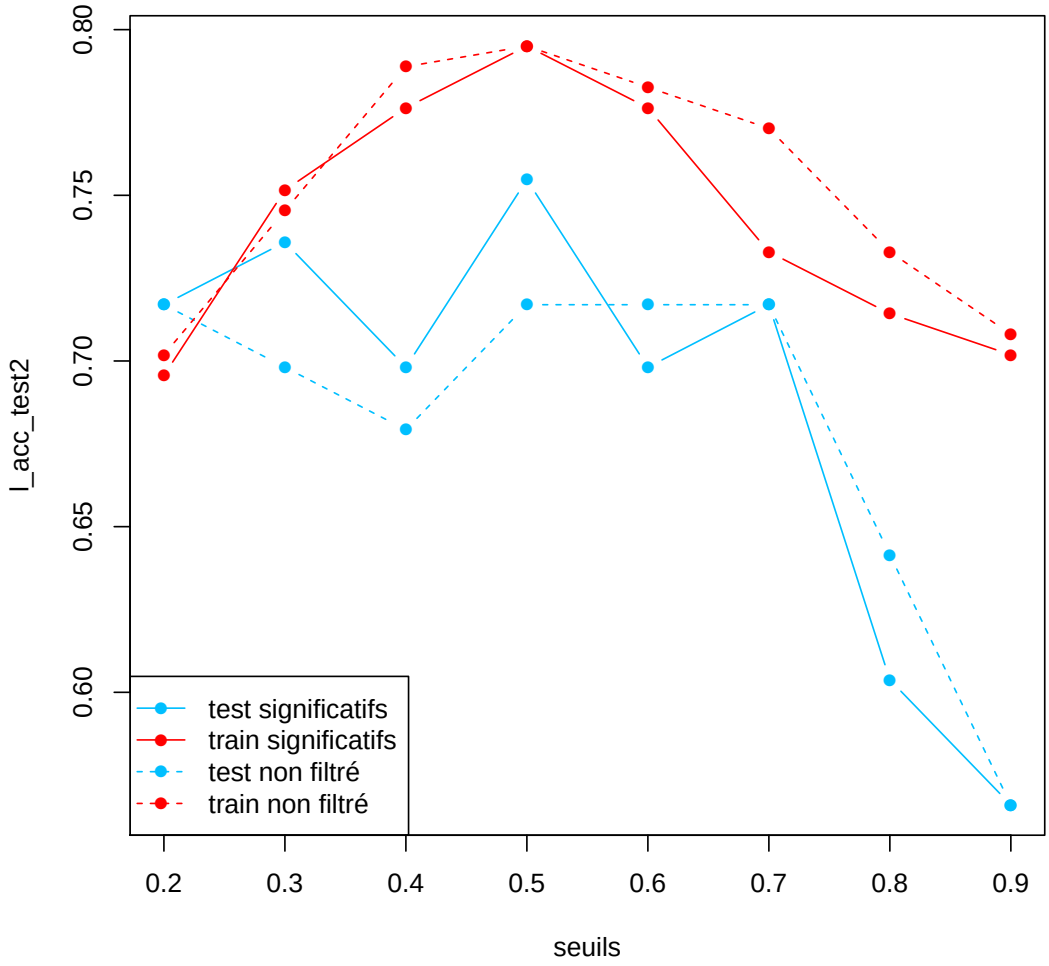
```
## [1] 0.754717
```

```
    proba_train <- predict(model2,newdata=train, type="response")
    print(sum((proba_train>0.5)== train$y)/ length(train$y))
```

```
## [1] 0.7950311
```

```
    seuils <- c(0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9)
    d_acc <- c()
    l_acc_train2 <- c()
    l_acc_test2 <- c()
    for (s in seuils){
      p_train <- proba_train > s
      acc_train <- sum(p_train== train$y)/ length(train$y)
      p_test <- proba_test > s
      acc_test <- sum(p_test== test$y)/ length(test$y)
      l_acc_test2 <- c(l_acc_test2, acc_test)
      l_acc_train2 <- c(l_acc_train2, acc_train)
    }

    all_pred <- c(all_pred, l_acc_test2, l_acc_train2)
    plot(seuils, l_acc_test2, type = "b", pch = 16, col ="deepskyblue",
         ylim = c(min(all_pred),max(all_pred)))
    lines(seuils, l_acc_train2,type = "b", pch = 16, col = "red")
    lines(seuils, l_acc_test,type = "b", pch = 16, col = "deepskyblue", lty = 2)
    lines(seuils, l_acc_train,type = "b", pch = 16, col = "red", lty =2)
    legend(x = "bottomleft",
       legend = c("test significatifs",
                  "train significatifs",
                  "test non filtré",
                  "train non filtré"),
       col = c("deepskyblue",
            "red",
            "deepskyblue",
            "red"),
       lty = c(1, 1, 2, 2),
       pch = 16)
```



Remarque 1. Pour améliorer la confiance que l'on a en les résultats observés et donc comparer plusieurs modèles, il faudrait normalement répéter les expériences plusieurs fois (10 fois en général) sur différents *splits* train et test, et calculer la valeur moyenne et l'écart-type sur ces 10 expériences.

Remarque 2. Il faudrait normalement *tuner* le seuil qui permet de prendre sa décision, *i.e.* affecter une données dans une classe ou une autre, il faudrait donc effectuer ce que l'on appelle une *cross-validation*.