

## Modèles Linéaires

### Correction Séance 5 Licence 3 MIAHS (2022-2023)

Guillaume Metzler, Francesco Amato  
Institut de Communication (ICOM)  
Université de Lyon, Université Lumière Lyon 2  
Laboratoire ERIC UR 3083, Lyon, France

[guillaume.metzler@univ-lyon2.fr](mailto:guillaume.metzler@univ-lyon2.fr) ; [francesco.amato@univ-lyon2.fr](mailto:francesco.amato@univ-lyon2.fr)

#### Résumé

Cette quatrième se focalise sur un modèle généralisé des modèles linéaires que l'on appelle la **régression logistique** :

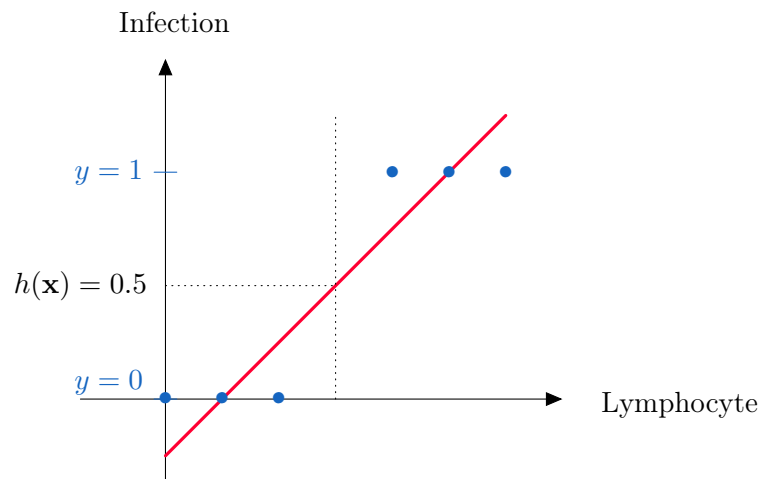
- Présentation de la régression logistique
- Estimation par maximum de vraisemblance
- Evaluation du modèle et applications

## 1 Vers l'intérêt de la régression logistique

Plaçons dans un contexte un peu différent de celui que nous avons traité jusqu'à présent, et considérons l'exemple suivant.

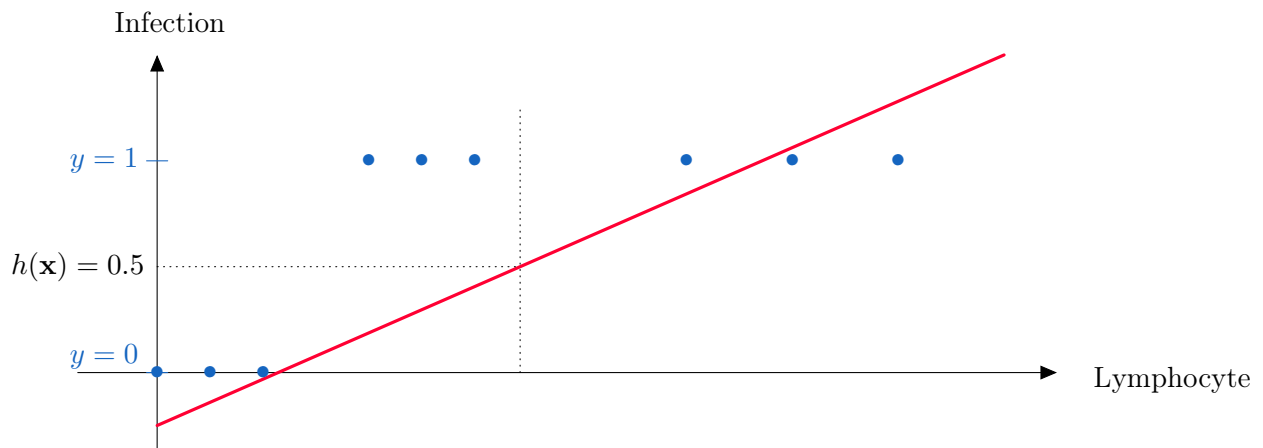
On cherche à construire un modèle de régression capable de déterminer si un individu est atteint ou non d'une infection en fonction de sa numération en lymphocytes. La variable prédite peut prendre deux valeurs : 1 si la personne a une infection et 0 sinon.

À première vue, rien ne nous empêche d'apprendre un modèle linéaire pour tenter d'ajuster notre nouveau nuage de points, comme illustré ci-dessous.



Il suffira alors de prendre un seuil, sur les valeurs prises par notre hypothèse  $h$ , au-delà duquel un individu sera considéré comme malade, *e.g.* on considère qu'un exemple  $\mathbf{x}$  appartient à la classe positive ( $y = 1$ ) lorsque l'hypothèse  $h$  renvoie une valeur supérieure à 0.5 (*i.e.* négatif sur la partie gauche et positif sur la partie droite). Dans cet exemple, cela fonctionne bien.

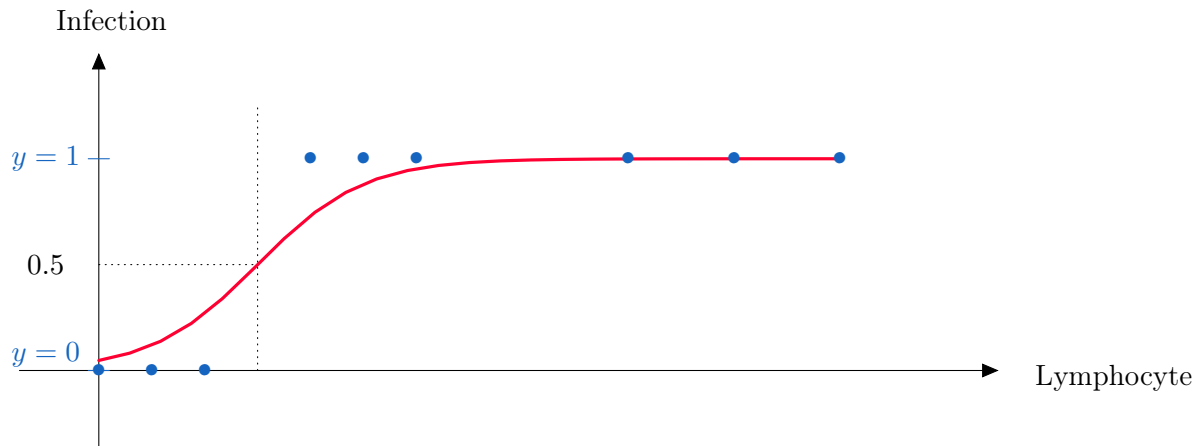
Considérons maintenant un autre cas où le nombre de lymphocytes peut être extrêmement élevé, ce qui signifie que l'infection est grave. Ce nouvel ensemble de données est représenté ci-dessous.



Cette fois-ci, nous constatons que si nous utilisons le même seuil, nous manquons des cas positifs ou des personnes infectées.

## 2 Vers la Régression Logistique

L'exemple précédent montre que la façon dont nous modélisons notre problème n'est pas bien choisie, nous avons besoin d'une structure différente, c'est-à-dire d'une courbe plus adaptée à la structure de nos données. Par exemple, nous avons besoin d'un modèle qui soit représenté comme suit :



Un tel modèle prend ses valeurs dans  $[0, 1]$  et nous pouvons donc dire qu'il estime la probabilité d'avoir une infection. Pour transformer les valeurs prédites par un modèle de régression linéaire en probabilités, nous utilisons la fonction logistique, *i.e.* nous calculons :

$$\frac{1}{1 + \exp(-\mathbf{x}\boldsymbol{\beta})}.$$

On parle alors de *Régression Logistique*.

**Régression Logistique : théorie et apprentissage** Le modèle de *Régression Logistique*, également appelé modèle *logit*, a été introduit au milieu du 20<sup>me</sup> siècle, mais l'utilisation des modèles *logit* remonte à la fin du 19<sup>me</sup> siècle.

Pour estimer la probabilité qu'un exemple appartienne à une classe donnée, par exemple la classe positive :  $\eta = Pr(Y = 1 | X)$ , la régression logistique vise à calculer le logarithme du *odds*, c'est-à-dire le rapport des probabilités. Nous estimons ensuite le logarithme de ce rapport à l'aide d'un modèle linéaire :

$$\ln \left( \frac{Pr(y = 1 | \mathbf{x})}{Pr(y = 0 | \mathbf{x})} \right) = \mathbf{x}\boldsymbol{\beta} + \varepsilon.$$

Ainsi, une fois les paramètres  $\boldsymbol{\beta}$  du modèle sont appris, nous pouvons calculer la probabilité d'appartenir à la classe 1 :

$$Pr(y = 1 \mid \mathbf{x}) = \frac{\exp(\mathbf{x}\boldsymbol{\beta})}{1 + \exp(\mathbf{x}\boldsymbol{\beta})} = \frac{1}{1 + \exp(-\boldsymbol{\beta}\mathbf{x})}.$$

Une telle fonction est appelée *fonction logistique* et prend ses valeurs dans  $[0, 1]$ . Un exemple  $\mathbf{x}_i$  est (généralement) prédit dans la classe 1 si  $Pr(y = 1 \mid \mathbf{x}) > 0.5$ , c'est-à-dire si  $\mathbf{x}\boldsymbol{\beta} > 0$ . Compte tenu d'une tâche et d'un objectif, nous pouvons choisir de modifier ce seuil.

Pour estimer les paramètres du modèle, nous maximisons la vraisemblance des données  $\mathcal{L}(\boldsymbol{\beta}, S)$ , où  $S = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m\}$  est un ensemble de  $m$  exemples.

$$\begin{aligned} \mathcal{L}(\boldsymbol{\beta}, S) &= \prod_{i=1}^m Pr(Y = y_i \mid X = \mathbf{x}_i), \\ &\quad \downarrow \text{on sépare } y_i = 0 \text{ et } y_i = 1 \\ &= \prod_{i=1, y_i=1}^m Pr(Y = y_i \mid X = \mathbf{x}_i) \times \prod_{i=1, y_i=0}^m Pr(Y = y_i \mid X = \mathbf{x}_i), \\ &\quad \downarrow \text{on utilise le fait que l'on suit une loi de Bernoulli} \\ &\quad \downarrow \text{nous n'avons que deux issues possibles} \\ &= \prod_{i=1}^m \left( \frac{1}{1 + \exp(-\mathbf{x}_i\boldsymbol{\beta})} \right)^{y_i} \times \left( \frac{1}{1 + \exp(\mathbf{x}_i\boldsymbol{\beta})} \right)^{(1-y_i)}. \end{aligned}$$

Notez que nous préférons généralement minimiser la log-vraisemblance négative des données :

$$\begin{aligned} \ell(\boldsymbol{\beta}, S) &= -\ln(\mathcal{L}(\boldsymbol{\beta}, S)), \\ &= -\sum_{i=1}^m y_i \ln \left( \frac{1}{1 + \exp(-\mathbf{x}_i\boldsymbol{\beta})} \right) + (1 - y_i) \ln \left( 1 - \frac{1}{1 + \exp(-\mathbf{x}_i\boldsymbol{\beta})} \right). \end{aligned}$$

Ce faisant, nous trouvons la fonction de perte logistique introduite précédemment. Dans ce qui suit, par souci de simplicité, nous fixerons  $g(\boldsymbol{\beta}, \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{x}\boldsymbol{\beta})}$ . On est donc ramené à résoudre le problème d'optimisation suivant

$$\min_{\boldsymbol{\beta} \in \mathbb{R}^{d+1}} -\frac{1}{m} \sum_{i=1}^m y_i \ln(g(\boldsymbol{\beta}, \mathbf{x}_i)) + (1 - y_i) \ln(1 - g(\boldsymbol{\beta}, \mathbf{x}_i)).$$

Nous divisons la perte par un facteur  $m$  afin d'être cohérent avec la notion de *moyenne des erreurs*

Par rapport au modèle linéaire de régression des moulinets, il n'existe pas de solutions analytiques. Cependant, le problème étant convexe, nous pouvons utiliser un algorithme basé sur le gradient pour trouver une solution.

**Apprentissage du modèle** On peut calculer le gradient de la fonction  $\ell$  par rapport au vecteur  $\beta$ . On a donc

$$\begin{aligned}\nabla \ell(\beta, S) &= \begin{bmatrix} \frac{\partial \ell}{\partial \beta_0}(\beta, S) \\ \frac{\partial \ell}{\partial \beta_1}(\beta, S) \\ \vdots \\ \frac{\partial \ell}{\partial \beta_d}(\beta, S) \end{bmatrix}, \\ &= - \sum_{i=1}^m -y_i(1 - g(\beta, \mathbf{x}_i))\mathbf{x}_i + (1 - y_i)g(\beta, \mathbf{x}_i)\mathbf{x}_i, \\ &= - \sum_{i=1}^n \left( y_i - \frac{1}{1 + \exp(-\mathbf{x}_i\beta)} \right) \mathbf{x}_i.\end{aligned}$$

Nous pouvons ensuite appliquer l'algorithme de descente de gradient en utilisant l'expression ci-dessus du gradient de la log-vraisemblance négative (on l'appellera fonction de coût) :

$$\beta^{(k+1)} = \beta^{(k)} - \eta \nabla \ell(\beta^{(k)}, S),$$

$k = 1, 2, \dots$  et  $\eta$  est le pas d'apprentissage.

La plupart du temps, nous utilisons l'algorithme de descente de gradient de **Newton-Raphson** pour minimiser notre fonction de coût, *i.e.* nous utilisons la matrice hessienne de  $\ell$  dans notre procédure de minimisation au lieu du pas d'apprentissage  $\eta$ .

Cette matrice hessienne est donnée par

$$\nabla^2 \ell(\beta, S) = \begin{bmatrix} \frac{\partial^2 \ell}{\partial \beta_0^2}(\beta, S) & \cdots & \frac{\partial^2 \ell}{\partial \beta_0 \partial \beta_d}(\beta, S) \\ \frac{\partial^2 \ell}{\partial \beta_1 \partial \beta_0}(\beta, S) & \cdots & \frac{\partial^2 \ell}{\partial \beta_1 \partial \beta_d}(\beta, S) \\ \vdots & & \vdots \\ \frac{\partial^2 \ell}{\partial \beta_d \partial \beta_0}(\beta, S) & \cdots & \frac{\partial^2 \ell}{\partial \beta_d^2}(\beta, S) \end{bmatrix} = \sum_{i=1}^m g(\beta, \mathbf{x}_i) (1 - g(\beta, \mathbf{x}_i)) \mathbf{x}_i \mathbf{x}_i^T.$$

On peut exprimer la matrice hessienne sous une forme plus compacte comme suit :

$$\nabla^2 \ell(\beta, \mathbf{X}) = \mathbf{X}^T \mathbf{G} \mathbf{X},$$

où  $\mathbf{X} \in \mathbb{R}^{m \times (d+1)}$  est la matrice de design (matrice *i.e.* des données) et  $\mathbf{G} \in \mathbb{R}^{m \times m}$  est la matrice définie par :

$$G = \begin{bmatrix} g(\beta, \mathbf{x}_1) (1 - g(\beta, \mathbf{x}_1)) & 0 & \cdots & \cdots & 0 \\ 0 & g(\beta, \mathbf{x}_2) (1 - g(\beta, \mathbf{x}_2)) & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & g(\beta, \mathbf{x}_m) (1 - g(\beta, \mathbf{x}_m)) \end{bmatrix}.$$

Notez qu'avec l'expression ci-dessus, la matrice hessienne est exprimée comme une combinaison linéaire positive de matrices de Gram et est donc une matrice semi-définie positive. L'algorithme de Newton-Raphson est donc le suivant :

$$\beta^{(k+1)} = \beta^{(k)} - \left( \nabla^2 \ell(\beta^{(k)}, S) \right)^{-1} \nabla \ell(\beta^{(k)}, S),$$

qui présente un taux de convergence plus rapide que l'algorithme standard de descente de gradient à pas constant.

### 3 Mise en pratique

Pour illustrer l'utilisation de la Régression Logistique, on se base sur la présentation effectuée [ici](#) et on se servira également du jeux de données *leukemia*

```
data = read.csv("data/leukemia.csv", sep=";")
head(data)
```

```
## REMISS CELL SMEAR INFIL LI BLAST TEMP
## 1      1  0.8  0.83  0.66 1.9  1.10 1.00
## 2      1  0.9  0.36  0.32 1.4  0.74 0.99
## 3      0  0.8  0.88  0.70 0.8  0.18 0.98
## 4      0  1.0  0.87  0.87 0.7  1.05 0.99
## 5      1  0.9  0.75  0.68 1.3  0.52 0.98
## 6      0  1.0  0.65  0.65 0.6  0.52 0.98
```

On s'intéresse ici la prédiction de la valeur de la variable rémission après traitement (**REMISS**) en fonction des différentes informations :

- **CELL** : Cellularité (pourcentage) du caillot de moelle,

- **SMEAR** : Pourcentage différentiel de blastes cancéreux,
- **INFIL** : Pourcentage d'infiltration de leucémie médullaire absolue,
- **LI** : Indice d'étiquetage en pourcentage des cellules leucémiques de la moelle osseuse (vis-à-vis autres cells),
- **BLAST** : Nombre absolu de blastes cancéreux, en milliers,
- **TEMP** : Température la plus élevée avant le début du traitement.

On cherche donc à savoir si un traitement administré à un patient fonctionnera ou non. On peut commencer par regarder la répartition de la variable que l'on cherche à prédire **REMISS**.

```
# La fonction "table", donne une table avec le nombre d'occurrences
# de l'objet fourni
prop = table(data$REMISS)
prop

##
##  0  1
## 18  9
```

Dans notre jeu de données, nous avons donc 9 patients qui sont entrés en rémission et 18 patients pour lesquels le traitement n'a pas fonctionné.

On va maintenant chercher à construire notre modèle qui nous permettra d'effectuer notre tâche prédictive et on testera ensuite son efficacité sur un jeu de données indépendant. Pour cela, on commence par séparer notre jeu de données en deux ensembles (Figure 1) : *train* et *test*, le premier nous servira à apprendre les paramètres  $\beta$  du modèle et le deuxième à tester notre modèle, *i.e.* voir si le modèle appris est capable de généraliser sur des données semblables.

Pour cela, on commence par séparer notre jeu de données en deux ensembles

```
# On fixe la graine, pour s'assurer que l'on obtienne bien des résultats identiques
set.seed(4)

# Partition du jeu de données avec la fonction

library(caret)

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

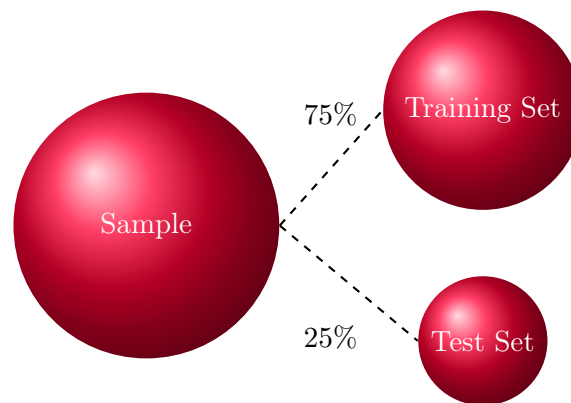



FIGURE 1 – Séparation du jeu de données en train/test avec une répartition 75/25

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

La fonction **createDataPartition** permet de créer une liste d'indice contenant 75% des exemples ici tout en conservant la proportion des classes en présence.

On peut maintenant apprendre notre modèle logistique à l'aide de la fonction **glm** de .

```
mymodel = glm(REMISS ~ ., data = train, family=binomial)
summary(mymodel)
```

```
##
## Call:
## glm(formula = REMISS ~ ., family = binomial, data = train)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -1.7929  -0.6075  -0.1383   0.1845   1.6706
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -127.025    172.241  -0.737   0.4608
## CELL        -14.523     22.082  -0.658   0.5107
## SMEAR       -30.502     34.482  -0.885   0.3764
## INFIL        41.887     39.068   1.072   0.2836
## LI           9.476      5.148   1.841   0.0657 .
## BLAST       -5.135      3.552  -1.446   0.1483
```



```
## TEMP          128.892    172.806    0.746    0.4557
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 26.734  on 20  degrees of freedom
## Residual deviance: 14.189  on 14  degrees of freedom
## AIC: 28.189
##
## Number of Fisher Scoring iterations: 6
```

Noter l'utilisation du paramètre "*family = binomial*" dans la fonction **glm**. L'argument binomial fait référence à la loi binomial et donc au fait que la valeur que l'on cherche à prédire ne peut prendre que deux modalités (0 ou 1).

La significativité des coefficients s'interprète de la même façon que pour le modèle linéaire classique. Ici on remarque que toutes les variables **ne sont pas** significatives sauf la variable **LI**. La qualité du modèle est évaluée selon le critère **AIC** qui est un critère analogue au **BIC** (noter qu'il existe un **AICc**, *i.e.* un **AIC** corrigé pour les échantillons de petites tailles). On

**Remarques :** cette analyse est à nuancer étant donnée le faible nombre d'exemples dont nous disposons pour effectuer cette analyse. Le critère **AIC** peut notamment être employé pour effectuer de la sélection de modèle, *i.e.* construire un modèle plus simple mais qui ne contient qu'un nombre restreint de variables.

```
mymodel_bis = glm(REMISS ~ CELL+LI, data=train, family=binomial)
summary(mymodel_bis)

##
## Call:
## glm(formula = REMISS ~ CELL + LI, family = binomial, data = train)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -2.0137  -0.5909  -0.3507   0.4643   1.5281
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -11.377      8.164  -1.393   0.1635
## CELL           6.704      7.549   0.888   0.3745
## LI            4.309      2.004   2.150   0.0315 *
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 26.734  on 20  degrees of freedom
## Residual deviance: 17.562  on 18  degrees of freedom
## AIC: 23.562
##
## Number of Fisher Scoring iterations: 6
```

Ce nouveau modèle a un **AIC** plus petit que le modèle complet et présente donc une qualité supérieure.

**Capacités prédictives du modèle** On cherche maintenant à savoir si notre modèle est capable d'effectuer de bonnes prédictions. On va commencer par regarder (i) s'il a réussi à apprendre quelque chose de nos données. On teste donc ses performances sur le train. Enfin, (ii) s'il est capable de généraliser et donc tester ses performances sur des données tests qu'il n'a pas rencontrées lors de l'apprentissage. On effectuera notre étude sur notre modèle réduit.

- **Train** : on regarde les prédictions sur le *train*

```
predictTrain = predict(mymodel_bis,newdata=train, type="response")

summary(predictTrain)
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.001374 0.071327 0.226925 0.333333 0.516728 0.934154
tapply(predictTrain, train$REMISS, mean)
##           0           1
## 0.2060604 0.5878791
```

- **Test** : on regarde les prédictions sur le *test*

```
predictTest = predict(mymodel_bis,newdata=test, type="response")

summary(predictTest)
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.007655 0.023762 0.285611 0.420586 0.849207 0.971087
tapply(predictTest, test$REMISS, mean)
##           0           1
## 0.2589253 0.7439072
```

On voit que les prédictions retournées sont des probabilités d'être en rémission. Par défaut, on suppose que si la probabilité d'être en rémission est plus grande que 0.5, alors le modèle prédit qu'il est plus probable d'avoir une rémission (que de ne pas en avoir).

Pour évaluer les performances du modèle, on va maintenant comparer les prédictions du modèles aux vraies valeurs. On va dresser ce que l'on appelle une *matrice de confusion*.

```
true_label = test$REMISS
predicted_label = 1*(predictTest>0.5)
res = table(actual = true_label, predict = predicted_label)
res

##          predict
## actual 0 1
##      0 3 1
##      1 0 2
```

La table peut se lire de la façon suivante : 2 rémission ont bien été retrouvés par le modèle, 1 individu a été prédit comme entré en rémission alors que ce n'est pas le cas et 3 individus qui ne sont pas en rémission ont bien été prédits comme n'étant pas entrés en rémission.

Bien évidemment, si on change le seuil pour l'assignation des classes, cette table de contingence (autre nom de la matrice de confusion) va aussi être modifiée.

```
true_label = test$REMISS
predicted_label = 1*(predictTest>0.2)
res_other = table(actual = true_label, predict = predicted_label)
res_other

##          predict
## actual 0 1
##      0 3 1
##      1 0 2
```

**Vers une analyse des performances** On peut commencer par évaluer les performances du modèle en évaluant son taux de bonne classification. Ce que l'on appelle aussi l'Accuracy.

```
acc = sum(diag(res)) / sum(res)
acc

## [1] 0.8333333
```

On peut aussi définir des mesures comme la spécificité ou encore la sensibilité du modèle. La *sensibilité* ou *rappel* est une quantité qui mesure la proportion de positifs (ici de rémission) retrouvé par le modèle. La *spécificité* est une quantité qui mesure la proportion de négatifs (ici de non-rémission) retrouvé par le modèle.

```
sensitivity = res[2,2]/sum(res[2,])
sensitivity

## [1] 1

specitivity = res[1,1]/sum(res[1,])
specitivity

## [1] 0.75
```

Il existe d'autres critères comme l'AUC ou tout simplement le tracé de la courbe ROC qui permettent d'avoir une idée des capacités prédictives locales et globales du modèle. Cette mesure est particulièrement adaptée lorsque le modèle retourne des scores d'appartenance à une classe. Mais nous reviendrons sur ces points là dans un cours de *Machine Learning*.