Tutoriel machine learning

Laurent Rouvière

24-25 juin 2021

Table des matières

Le tutoriel utilise le packages suivants :

```
library(tidyverse)
library(tidymodels)
library(car)
library(bestglm)
library(glmnet)
library(kernlab)
library(rpart)
library(rpart.plot)
library(ranger)
library(pROC)
```

Régression

On considère le jeu de données `ozone.txt`

```
ozone <- read.table("ozone.txt")
summary(ozone)
##      maxO3             T9             T12             T15       
##  Min.   : 42.00   Min.   :11.30   Min.   :14.00   Min.   :14.90  
##  1st Qu.: 70.75   1st Qu.:16.20   1st Qu.:18.60   1st Qu.:19.27  
##  Median : 81.50   Median :17.80   Median :20.55   Median :22.05  
##  Mean   : 90.30   Mean   :18.36   Mean   :21.53   Mean   :22.63  
##  3rd Qu.:106.00   3rd Qu.:19.93   3rd Qu.:23.55   3rd Qu.:25.40  
##  Max.   :166.00   Max.   :27.00   Max.   :33.50   Max.   :35.50  
##       Ne9             Ne12            Ne15            Vx9         
##  Min.   :0.000   Min.   :0.000   Min.   :0.00   Min.   :-7.8785  
##  1st Qu.:3.000   1st Qu.:4.000   1st Qu.:3.00   1st Qu.:-3.2765  
##  Median :6.000   Median :5.000   Median :5.00   Median :-0.8660  
##  Mean   :4.929   Mean   :5.018   Mean   :4.83   Mean   :-1.2143  
```

```
##  3rd Qu.:7.000   3rd Qu.:7.000   3rd Qu.:7.00   3rd Qu.: 0.6946
##  Max.   :8.000   Max.   :8.000   Max.   :8.00   Max.   : 5.1962
##       Vx12            Vx15           maxO3v          vent
##  Min.   :-7.878   Min.   :-9.000   Min.   : 42.00   Length:112
##  1st Qu.:-3.565   1st Qu.:-3.939   1st Qu.: 71.00   Class :character
##  Median :-1.879   Median :-1.550   Median : 82.50   Mode  :character
##  Mean   :-1.611   Mean   :-1.691   Mean   : 90.57
##  3rd Qu.: 0.000   3rd Qu.: 0.000   3rd Qu.:106.00
##  Max.   : 6.578   Max.   : 5.000   Max.   :166.00
##     pluie
##  Length:112
##  Class :character
##  Mode  :character
##
##
##
```

où le problème est d'expliquer la concentration quotidienne maximale en ozone (`maxO3`) par 12 autres variables.

1. Construire le modèle linéaire complet (avec toutes les variables explicatives).
   Il suffit d'utiliser la fonction `lm` :
   ```
   mod.complet <- lm(maxO3~.,data=ozone)
   ```

2. Expliquer comment les variables qualitatives (`vent` et `pluie`) sont prises en compte.
   Ces variables sont codées en indicatrices, par exemple pour `vent` on a

   $$Y = \cdots + \beta_E \mathbf{1}_{\text{vent=Est}} + \beta_N \mathbf{1}_{\text{vent=Nord}} + \beta_O \mathbf{1}_{\text{vent=Ouest}} + \beta_S \mathbf{1}_{\text{vent=Sud}} + \ldots$$

   Le modèle écrit ainsi n'étant pas identifiable, une contrainte identifiante est ajoutée. Par défaut, `R` fixe à 0 le coefficient associé à la première modalité. On a donc ici $\beta_E = 0$.

3. Faire un `summary` du modèle et expliquer la sortie.
   ```
   summary(mod.complet)
   ##
   ## Call:
   ## lm(formula = maxO3 ~ ., data = ozone)
   ##
   ## Residuals:
   ##     Min      1Q  Median      3Q     Max
   ## -51.814  -8.695  -1.020   7.891  40.046
   ##
   ## Coefficients:
   ##             Estimate Std. Error t value Pr(>|t|)
   ## (Intercept) 16.26536   15.94398   1.020   0.3102
   ## T9           0.03917    1.16496   0.034   0.9732
   ## T12          1.97257    1.47570   1.337   0.1844
   ## T15          0.45031    1.18707   0.379   0.7053
   ## Ne9         -2.10975    0.95985  -2.198   0.0303 *
   ## Ne12        -0.60559    1.42634  -0.425   0.6721
   ## Ne15        -0.01718    1.03589  -0.017   0.9868
   ## Vx9          0.48261    0.98762   0.489   0.6262
   ## Vx12         0.51379    1.24717   0.412   0.6813
   ## Vx15         0.72662    0.95198   0.763   0.4471
   ## maxO3v       0.34438    0.06699   5.141 1.42e-06 ***
   ## ventNord     0.53956    6.69459   0.081   0.9359
   ```

```
## ventOuest     5.53632    8.24792   0.671   0.5037
## ventSud       5.42028    7.16180   0.757   0.4510
## pluieSec      3.24713    3.48251   0.932   0.3534
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 14.51 on 97 degrees of freedom
## Multiple R-squared:  0.7686, Adjusted R-squared:  0.7352
## F-statistic: 23.01 on 14 and 97 DF,  p-value: < 2.2e-16
```

On obtient un tableau à 4 colonnes :

— **Estimate** : les estimations
— **Std. Error** : les écarts-types estimés des estimateurs
— **t value** : la statistique du test de nullité du paramètre
— **Pr(>|t|)** : la probabilité critique de ce test.

4. Comment juger de la pertinence de la variable `vent` dans ce modèle ?

   Dire que le vent n'a pas d'importance revient à dire que la concentration en ozone ne change pas lorsque la direction du vent change. Cela signifie que tous les coefficients associés à la variable `vent` dans le modèle sont identiques, ou encore, comte tenu de la contrainte :

$$\beta_N = \beta_O = \beta_S = 0.$$

   Il faut donc tester cette hypothèse contre sa négation. Un tel test peut s'effectuer à l'aide de la fonction `Anova` du package `car` :

```
Anova(mod.complet)
## Anova Table (Type II tests)
##
## Response: max03
##             Sum Sq Df F value    Pr(>F)
## T9             0.2  1  0.0011   0.97325
## T12          376.0  1  1.7868   0.18445
## T15           30.3  1  0.1439   0.70526
## Ne9         1016.5  1  4.8312   0.03033 *
## Ne12          37.9  1  0.1803   0.67208
## Ne15           0.1  1  0.0003   0.98680
## Vx9           50.2  1  0.2388   0.62619
## Vx12          35.7  1  0.1697   0.68127
## Vx15         122.6  1  0.5826   0.44715
## max03v      5560.4  1 26.4261 1.421e-06 ***
## vent         297.8  3  0.4718   0.70267
## pluie        182.9  1  0.8694   0.35344
## Residuals 20410.2 97
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

5. On souhaite maintenant sélectionner les variables. À l'aide du package `bestglm`, effectuer une procédure exhaustive en utilisant les critères **AIC** et **BIC**.

   Il faut au préalable mettre les données au bon format pour `bestglm` :

```
ozone1 <- ozone[,c(2:13,1)]
ozone1$vent <- as.factor(ozone1$vent)
ozone1$pluie <- as.factor(ozone1$pluie)
sel.BIC <- bestglm(ozone1)
sel.AIC <- bestglm(ozone1,IC="AIC")
```

   Les critères sélectionnent ici les mêmes variables :

```
sel.BIC$BestModel
##
## Call:
## lm(formula = y ~ ., data = data.frame(Xy[, c(bestset[-1], FALSE),
##     drop = FALSE], y = y))
##
## Coefficients:
## (Intercept)          T12          Ne9          Vx9       maxO3v
##     12.6313       2.7641      -2.5154       1.2929       0.3548
sel.AIC$BestModel
##
## Call:
## lm(formula = y ~ ., data = data.frame(Xy[, c(bestset[-1], FALSE),
##     drop = FALSE], y = y))
##
## Coefficients:
## (Intercept)          T12          Ne9          Vx9       maxO3v
##     12.6313       2.7641      -2.5154       1.2929       0.3548
```

On peut également visualiser les groupes de tête

```
sel.BIC$BestModels
##      T9  T12   T15  Ne9  Ne12  Ne15   Vx9  Vx12  Vx15 maxO3v  vent pluie
## 1 FALSE TRUE FALSE TRUE FALSE FALSE  TRUE FALSE FALSE   TRUE FALSE FALSE
## 2 FALSE TRUE FALSE TRUE FALSE FALSE FALSE FALSE FALSE   TRUE FALSE FALSE
## 3 FALSE TRUE FALSE TRUE FALSE FALSE FALSE  TRUE FALSE   TRUE FALSE FALSE
## 4 FALSE TRUE FALSE TRUE FALSE FALSE FALSE FALSE  TRUE   TRUE FALSE FALSE
## 5 FALSE TRUE FALSE TRUE FALSE FALSE FALSE FALSE FALSE   TRUE FALSE  TRUE
##   Criterion
## 1  604.8984
## 2  604.9037
## 3  606.3453
## 4  606.4493
## 5  608.4072
sel.AIC$BestModels
##      T9  T12   T15  Ne9  Ne12  Ne15   Vx9  Vx12  Vx15 maxO3v  vent pluie
## 1 FALSE TRUE FALSE TRUE FALSE FALSE  TRUE FALSE FALSE   TRUE FALSE FALSE
## 2 FALSE TRUE FALSE TRUE FALSE FALSE FALSE  TRUE FALSE   TRUE FALSE FALSE
## 3 FALSE TRUE FALSE TRUE FALSE FALSE FALSE FALSE  TRUE   TRUE FALSE FALSE
## 4 FALSE TRUE FALSE TRUE FALSE FALSE  TRUE FALSE FALSE   TRUE FALSE  TRUE
## 5 FALSE TRUE FALSE TRUE FALSE FALSE  TRUE FALSE  TRUE   TRUE FALSE FALSE
##   Criterion
## 1  594.0244
## 2  595.4713
## 3  595.5753
## 4  595.6001
## 5  595.6278
```

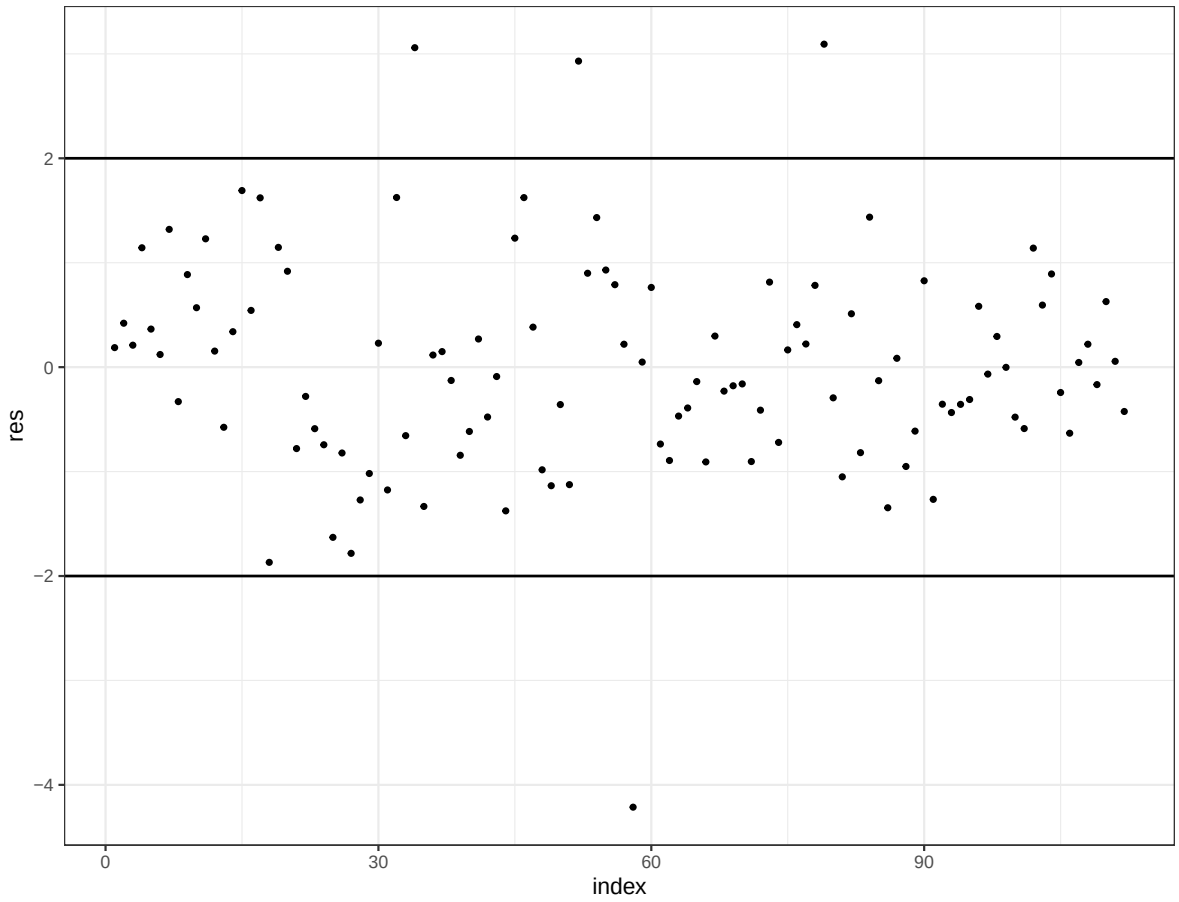
On récupère le modèle final :

```
final <- sel.BIC$BestModel
summary(final)
##
## Call:
## lm(formula = y ~ ., data = data.frame(Xy[, c(bestset[-1], FALSE),
##     drop = FALSE], y = y))
##
```

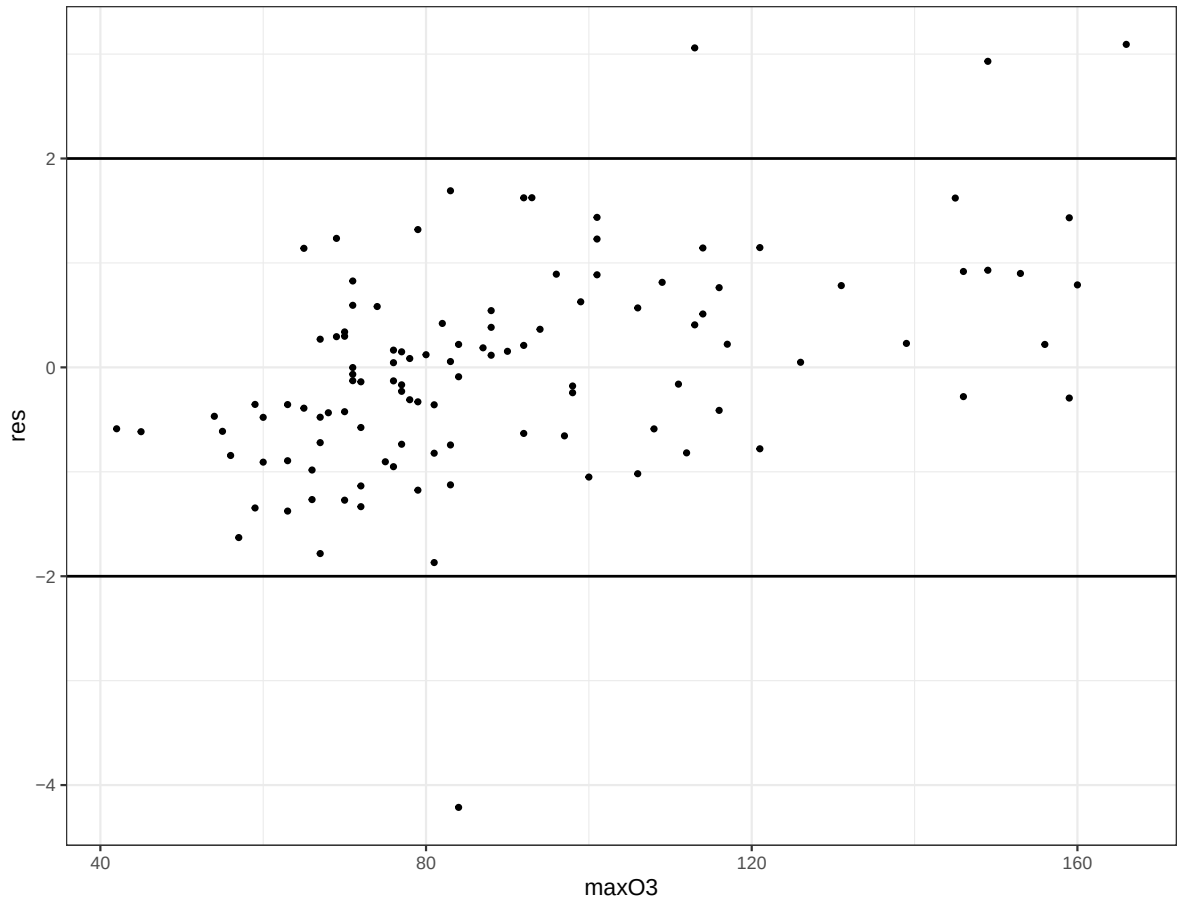
```
## Residuals:
##     Min      1Q  Median      3Q     Max
## -52.396  -8.377  -1.086   7.951  40.933
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 12.63131   11.00088   1.148 0.253443
## T12          2.76409    0.47450   5.825 6.07e-08 ***
## Ne9         -2.51540    0.67585  -3.722 0.000317 ***
## Vx9          1.29286    0.60218   2.147 0.034055 *
## max03v       0.35483    0.05789   6.130 1.50e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 14 on 107 degrees of freedom
## Multiple R-squared:  0.7622, Adjusted R-squared:  0.7533
## F-statistic: 85.75 on 4 and 107 DF,  p-value: < 2.2e-16
```

6. Visualiser les résidus studentisés du modèle final.

```
tbl <- tibble(index=1:nrow(ozone),res=rstudent(final),max03=ozone$max03)
ggplot(tbl)+aes(x=index,y=res)+geom_point()+geom_hline(yintercept = c(-2,2))
```




```
ggplot(tbl)+aes(x=max03,y=res)+geom_point()+geom_hline(yintercept = c(-2,2))
```



Régression logistique

On considère les données `SAheart` :

```
data(SAheart)
summary(SAheart)
##       sbp             tobacco            ldl              adiposity
##  Min.   :101.0   Min.   : 0.0000   Min.   : 0.980   Min.   : 6.74
##  1st Qu.:124.0   1st Qu.: 0.0525   1st Qu.: 3.283   1st Qu.:19.77
##  Median :134.0   Median : 2.0000   Median : 4.340   Median :26.11
##  Mean   :138.3   Mean   : 3.6356   Mean   : 4.740   Mean   :25.41
##  3rd Qu.:148.0   3rd Qu.: 5.5000   3rd Qu.: 5.790   3rd Qu.:31.23
##  Max.   :218.0   Max.   :31.2000   Max.   :15.330   Max.   :42.49
##     famhist        typea          obesity         alcohol           age
##  Absent :270   Min.   :13.0   Min.   :14.70   Min.   :  0.00   Min.   :15.00
##  Present:192   1st Qu.:47.0   1st Qu.:22.98   1st Qu.:  0.51   1st Qu.:31.00
##                Median :53.0   Median :25.80   Median :  7.51   Median :45.00
##                Mean   :53.1   Mean   :26.04   Mean   : 17.04   Mean   :42.82
##                3rd Qu.:60.0   3rd Qu.:28.50   3rd Qu.: 23.89   3rd Qu.:55.00
##                Max.   :78.0   Max.   :46.58   Max.   :147.19   Max.   :64.00
##       chd
##  Min.   :0.0000
##  1st Qu.:0.0000
##  Median :0.0000
##  Mean   :0.3463
```

```
##  3rd Qu.:1.0000
##  Max.   :1.0000
```

Sélection de variables

1. Construire le modèle logistique permettant d'expliquer `chd` par les autres variables.

```
mod.complet <- glm(chd~.,data=SAheart,family="binomial")
summary(mod.complet)
##
## Call:
## glm(formula = chd ~ ., family = "binomial", data = SAheart)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7781  -0.8213  -0.4387   0.8889   2.5435
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -6.1507209  1.3082600  -4.701 2.58e-06 ***
## sbp             0.0065040  0.0057304   1.135 0.256374
## tobacco         0.0793764  0.0266028   2.984 0.002847 **
## ldl             0.1739239  0.0596617   2.915 0.003555 **
## adiposity       0.0185866  0.0292894   0.635 0.525700
## famhistPresent  0.9253704  0.2278940   4.061 4.90e-05 ***
## typea           0.0395950  0.0123202   3.214 0.001310 **
## obesity        -0.0629099  0.0442477  -1.422 0.155095
## alcohol         0.0001217  0.0044832   0.027 0.978350
## age             0.0452253  0.0121298   3.728 0.000193 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 596.11  on 461  degrees of freedom
## Residual deviance: 472.14  on 452  degrees of freedom
## AIC: 492.14
##
## Number of Fisher Scoring iterations: 5
```

2. On considère le (faux) nouvel individu

```
(xnew <- SAheart[50,1:9])
##    sbp tobacco  ldl adiposity famhist typea obesity alcohol age
## 50 126     3.8 3.88     31.79  Absent    57   30.53       0  30
```

Estimer la probabilité $\mathbf{P}(chd = 1|X = x)$ pour ce nouvel individu. On pourra utiliser la fonction `predict`.

```
predict(mod.complet,newdata=xnew,type="response")
##        50
## 0.1119621
```

3. Effectuer une procédure de sélection de variable.

On choisit une procédure exhaustive par **BIC** :

```
sel.BIC <- bestglm(SAheart,family=binomial)
sel.BIC$BestModel %>% summary()
##
```

```
## Call:
## glm(formula = y ~ ., family = family, data = Xi, weights = weights)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9165  -0.8054  -0.4430   0.9329   2.6139
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -6.44644    0.92087  -7.000 2.55e-12 ***
## tobacco         0.08038    0.02588   3.106  0.00190 **
## ldl             0.16199    0.05497   2.947  0.00321 **
## famhistPresent  0.90818    0.22576   4.023 5.75e-05 ***
## typea           0.03712    0.01217   3.051  0.00228 **
## age             0.05046    0.01021   4.944 7.65e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 596.11  on 461  degrees of freedom
## Residual deviance: 475.69  on 456  degrees of freedom
## AIC: 487.69
##
## Number of Fisher Scoring iterations: 5
```

Courbe ROC

Il s'agit d'un critère fréquemment utilisé pour mesurer la performance d'un **score**. Étant donné (X, Y) un couple aléatoire à valeurs dans $\mathcal{X} \times \{-1, 1\}$, on rappelle qu'un score est une fonction $S : \mathcal{X} \to \mathbb{R}$. Dans la plupart des cas, un score s'obtient en estimant la probabilité $\mathbf{P}(Y = 1|X = x)$. Pour un seuil $s \in \mathbb{R}$ fixé, un score possède deux types d'erreur

$$\alpha(s) = \mathbf{P}(S(X) \geq s|Y = -1) \quad \text{et} \quad \beta(s) = \mathbf{P}(S(X) < s|Y = 1).$$

La courbe ROC est la **courbe paramétrée** définie par :

$$\begin{cases} x(s) = \alpha(s) = \mathbf{P}(S(X) > s|Y = -1) \\ y(s) = 1 - \beta(s) = \mathbf{P}(S(X) \geq s|Y = 1) \end{cases}$$

Elle permet de visualiser sur une seul graphe 2D ces deux erreurs pour toutes les valeurs de seuil s.

On sépare les données en un échantillon d'apprentissage et un échantillon test :

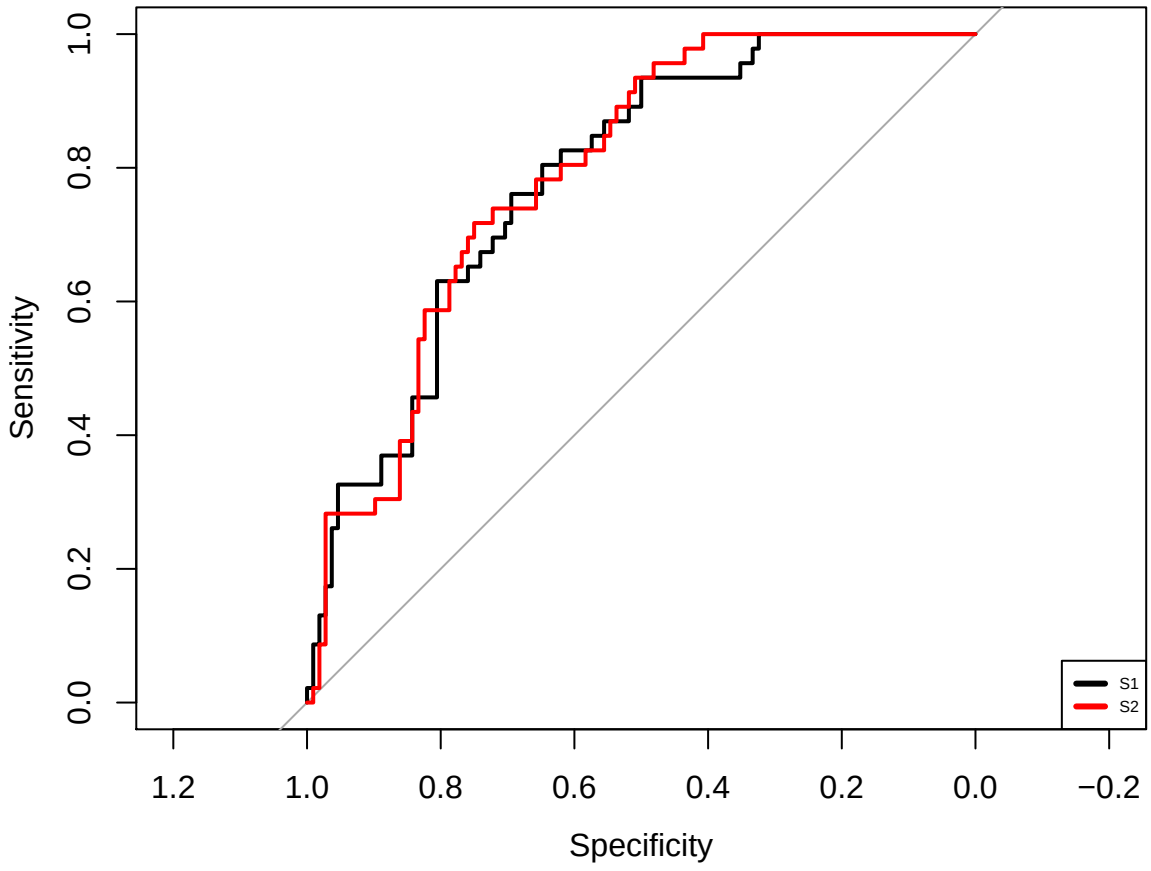
```
set.seed(123)
don_split <- initial_split(SAheart,prop=2/3)
dapp <- training(don_split)
dtest <- testing(don_split)
```

1. Construire le modèle logistique complet et celui qui contient les variables sélectionnés à la partie précédente sur les **données d'apprentissage uniquement**.
   ```
   complet <- glm(chd~.,data=dapp,family=binomial)
   sel <- glm(chd~tobacco + ldl + famhist + typea + age,data=dapp,family=binomial)
   ```
2. Calculer les probabilités $\mathbf{P}(chd = 1|X = x)$ estimées par ces deux modèles sur les individus de l'échantillon test.

```
tbl.prev <- tibble(complet=predict(complet,newdata=dtest,type="response"),
                   sel=predict(sel,newdata=dtest,type="response"),
                   obs=as.factor(dtest$chd))
```

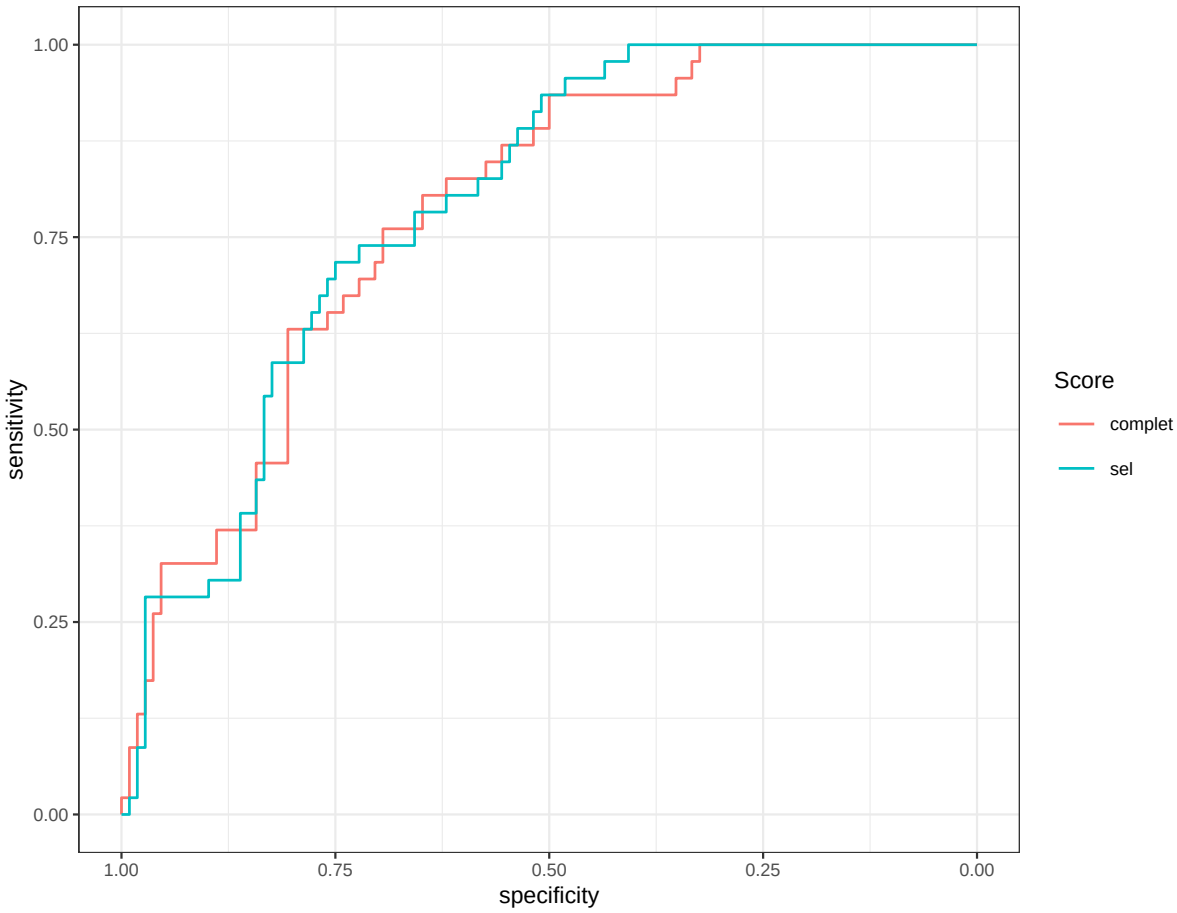
3. Visualiser les courbes ROC de ces deux modèles à l'aide du package `pROC`.

```
roc.obj <- roc(obs~.,data=tbl.prev)
plot(roc.obj[[1]])
plot(roc.obj[[2]],add=TRUE,col="red")
legend("bottomright",legend=c("S1","S2"),col=c("black","red"),lwd=3,cex=0.5)
```



Les courbes ggplot s'obtiennent avec :

```
pROC::ggroc(roc.obj)+labs(color="Score")
```



On obtient les **AUC** avec :

```
lapply(roc.obj,auc)
## $complet
## Area under the curve: 0.7842
##
## $sel
## Area under the curve: 0.7911
```

ou encore avec la syntaxe `tidy` :

```
tbl.prev %>%
  summarize_at(1:2,~roc_auc_vec(truth=obs,estimate=.,event_level="second"))
## # A tibble: 1 x 2
##   complet   sel
##     <dbl> <dbl>
## 1   0.784 0.791
```

Régularisation

On considère ici les données `spam` que l'on sépare en un échantillon d'apprentissage et un échantillon test :

```
data(spam)
set.seed(123)
spam_split <- initial_split(spam,prop=2/3)
dapp <- training(spam_split)
dtest <- testing(spam_split)
```

Le problème est d'expliquer la variable binaire `type` par les autres variables. On crée un `tibble` où on stockera les estimations des probabilités $\mathbf{P}(Y = spam|X = x)$ des algorithmes **ridge** et **lasso** :

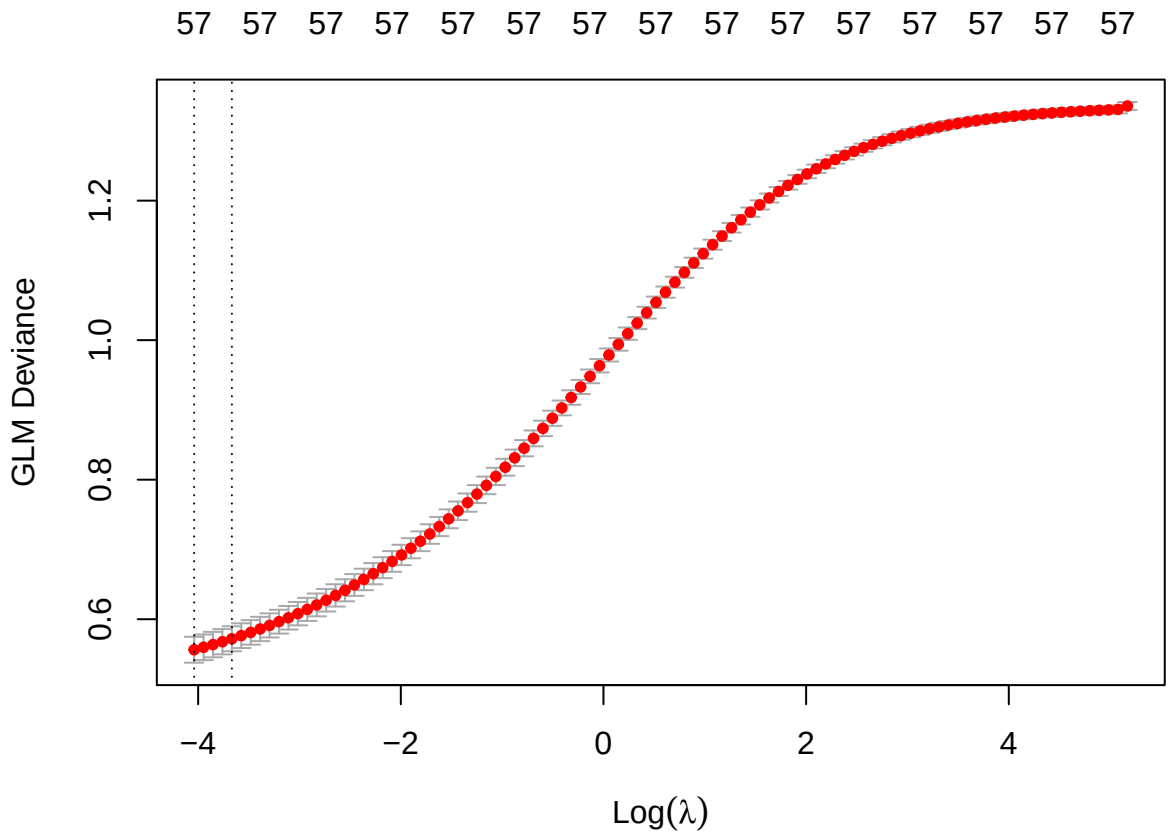
```
tbl.prev <- matrix(0,ncol=2,nrow=nrow(dtest)) %>% as_tibble()
names(tbl.prev) <- c("Ridge","Lasso")
```

On rappelle que `glmnet` n'admet pas de formule, il faut expliciter la matrice des `X` et le vecteur des `Y`.

```
X.app <- model.matrix(type~.,data=dapp)[,-1]
Y.app <- dapp$type
X.test <- model.matrix(type~.,data=dtest)[,-1]
Y.test <- dtest$type
```

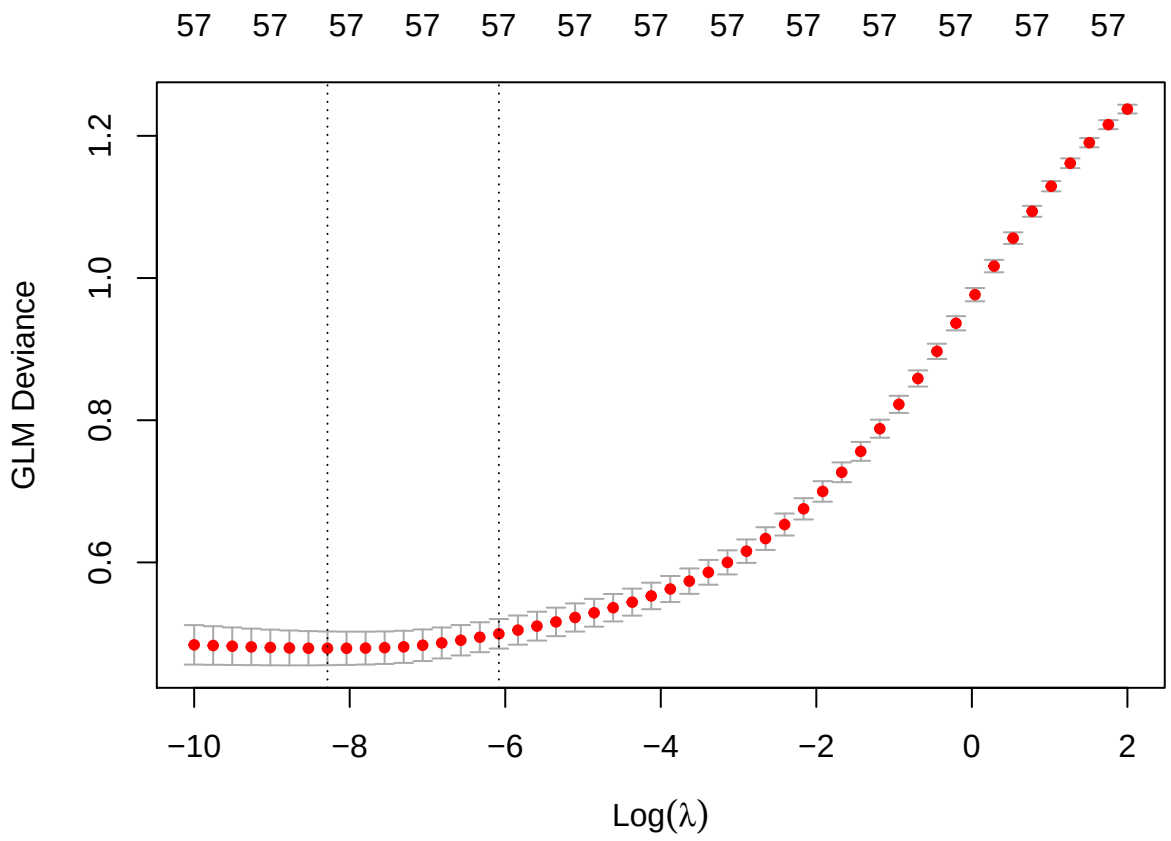
1. Entraîner l'algorithme `ridge` sur `dapp` et compléter la colonne `Ridge` de `tbl.prev`.
   On sélectionne `lambda` par validation croisée.

```
set.seed(123)
ridge.cv <- cv.glmnet(X.app,Y.app,alpha=0,family=binomial)
plot(ridge.cv)
```



La meilleure valeur se trouve à l'extrémité de la grille : il faut changer les valeurs par défaut :

```
set.seed(123)
ridge.cv <- cv.glmnet(X.app,Y.app,alpha=0,family=binomial,
                      lambda=exp(seq(-10,2,length=50)))
plot(ridge.cv)
```

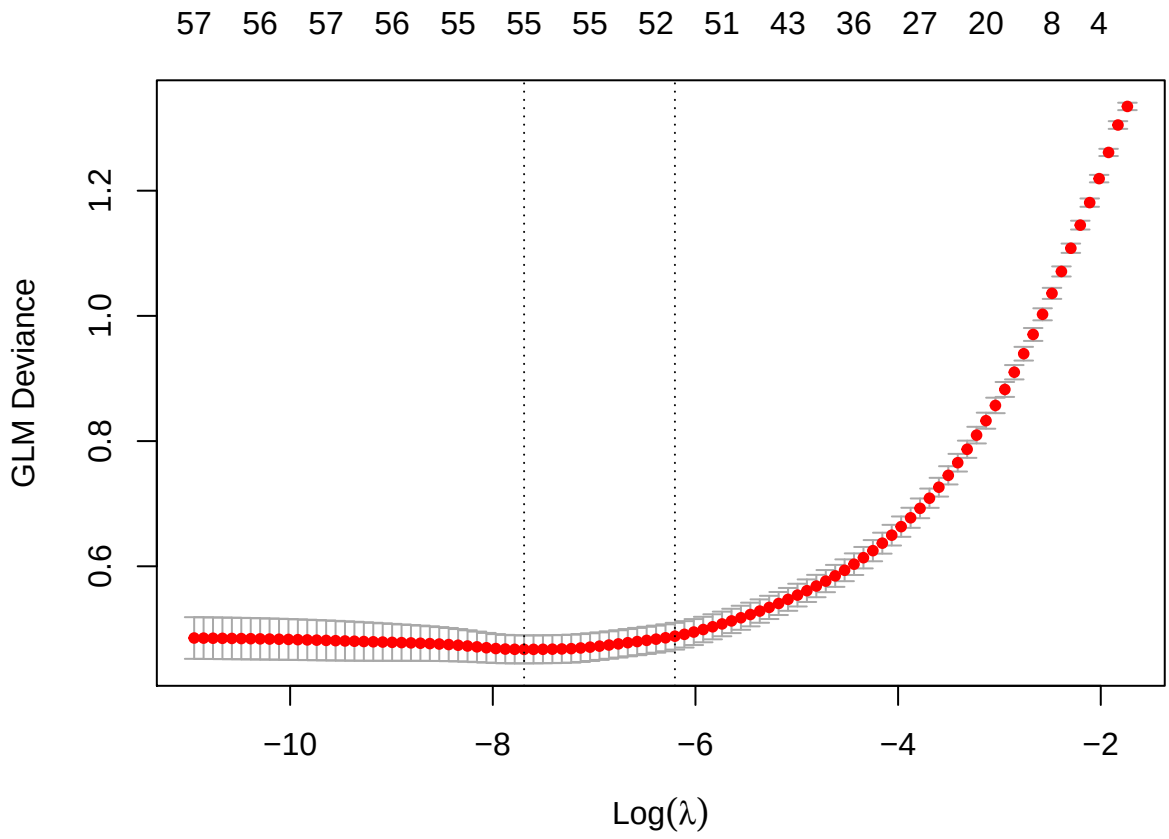



```
prev.ridge <- predict(ridge.cv,newx=X.test,type="response") %>% as.numeric()
tbl.prev$Ridge <- prev.ridge
```

2. Faire la même chose pour le lasso.

   On sélectionne `lambda` par validation croisée.

```
set.seed(123)
lasso.cv <- cv.glmnet(X.app,Y.app,alpha=1,family=binomial)
plot(lasso.cv)
```

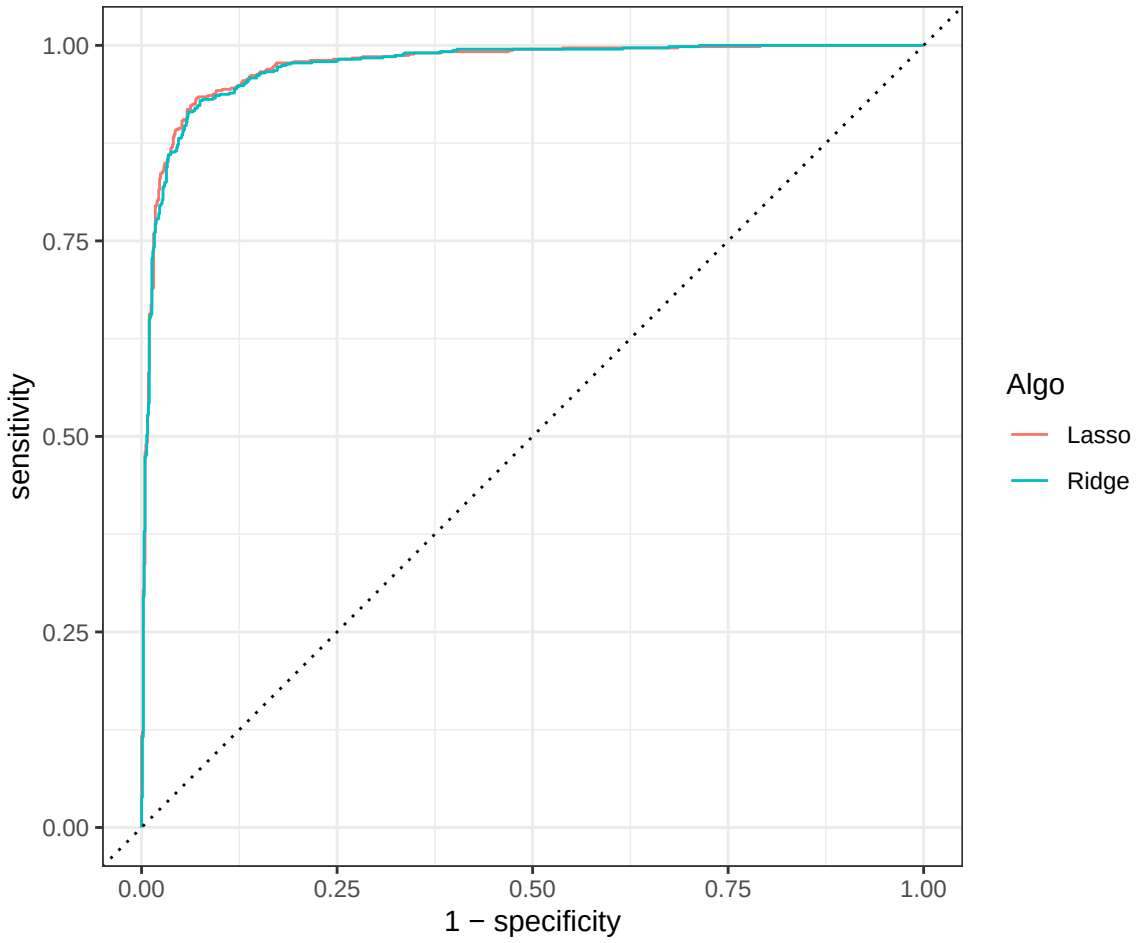


Ici tout est OK, on peut calculer les prévisions :

```
prev.lasso <- predict(lasso.cv,newx=X.test,type="response") %>% as.numeric()
tbl.prev$Lasso <- prev.lasso
```

3. Tracer les courbes ROC et calculer les AUC pour ces deux algorithmes.

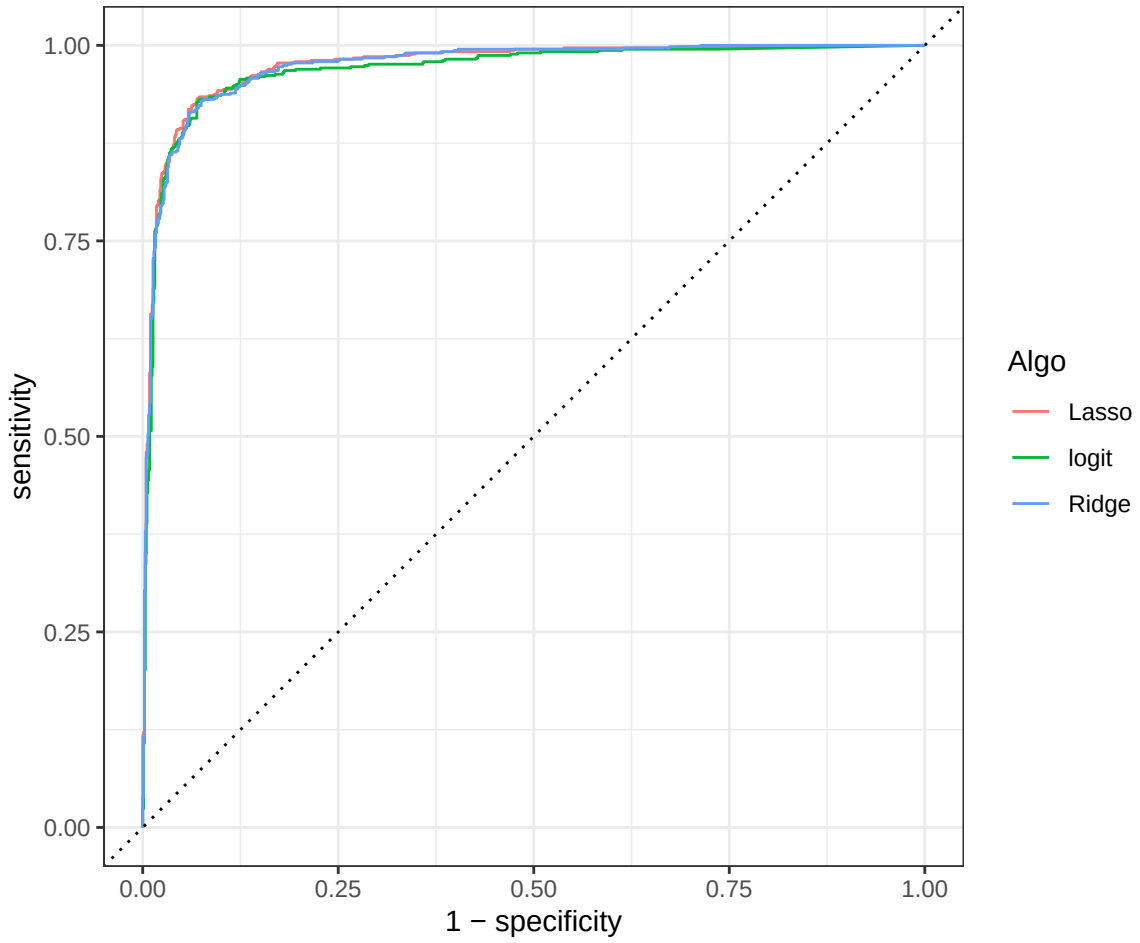
```
tbl.prev %>% mutate(obs=dtest$type) %>%
  pivot_longer(-obs,names_to="Algo",values_to = "score") %>%
  group_by(Algo) %>%
  roc_curve(truth=obs,estimate=score,event_level="second") %>%
  autoplot()
```




```
tbl.prev %>% mutate(obs=dtest$type) %>%
  summarize_at(1:2,~roc_auc_vec(truth=obs,estimate=.,event_level="second"))
## # A tibble: 1 x 2
##   Ridge Lasso
##   <dbl> <dbl>
## 1 0.974 0.975
```

On termine en comparant avec une régression logisique standard :

```
logit <- glm(type~.,data=dapp,family=binomial)
tbl.prev1 <- tbl.prev %>% mutate(logit=predict(logit,newdata=dtest,type="response"))
tbl.prev1 %>% mutate(obs=dtest$type) %>%
  pivot_longer(-obs,names_to="Algo",values_to = "score") %>%
  group_by(Algo) %>%
  roc_curve(truth=obs,estimate=score,event_level="second") %>%
  autoplot()
```



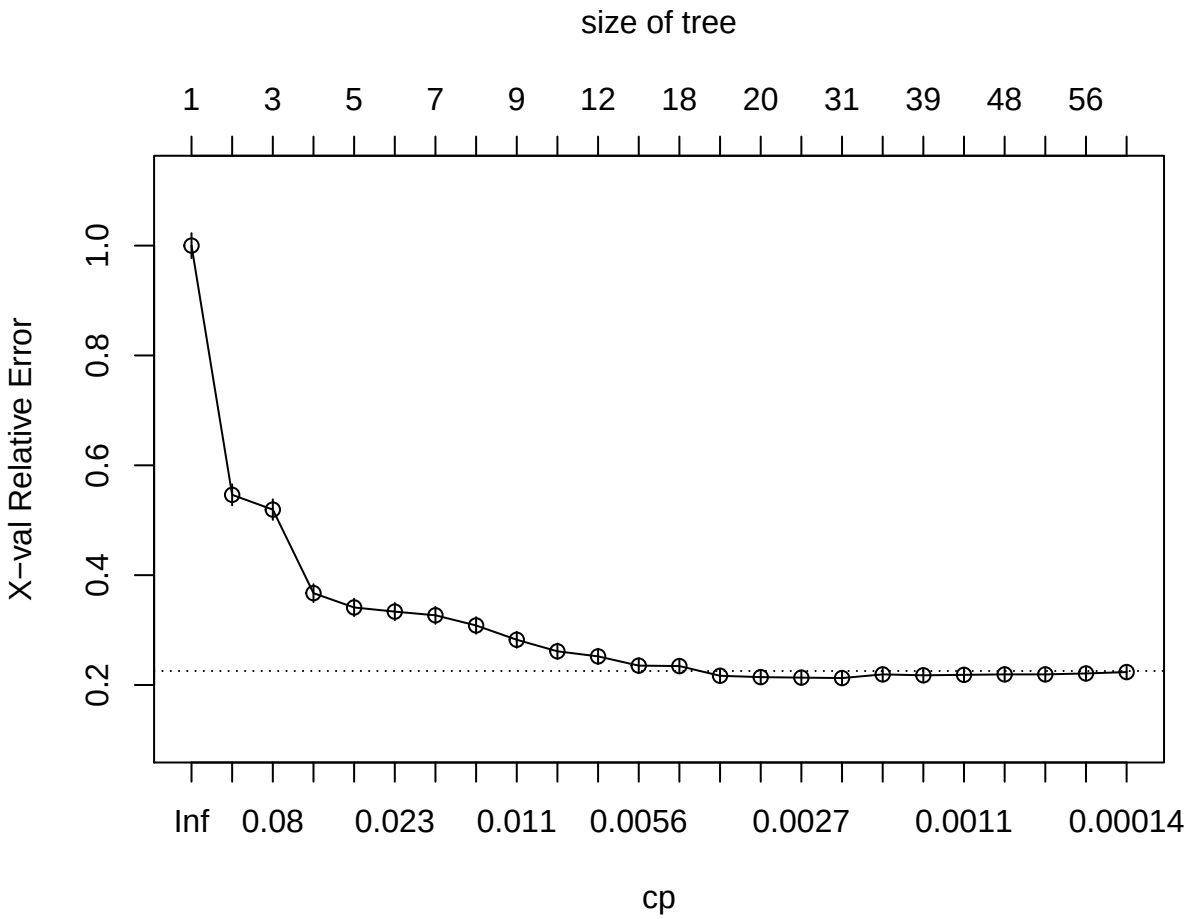

```r
tbl.prev1 %>% mutate(obs=dtest$type) %>%
  summarize_at(1:3,~roc_auc_vec(truth=obs,estimate=.,event_level="second"))
## # A tibble: 1 x 3
##   Ridge Lasso logit
##   <dbl> <dbl> <dbl>
## 1 0.974 0.975 0.969
```

Comparaison de méthodes

Sur les mêmes données que dans la partie précédente, construire un arbre CART et une forêt aléatoire. Comparer les performances en utilisant la courbe ROC (et d'autres critères si possible).

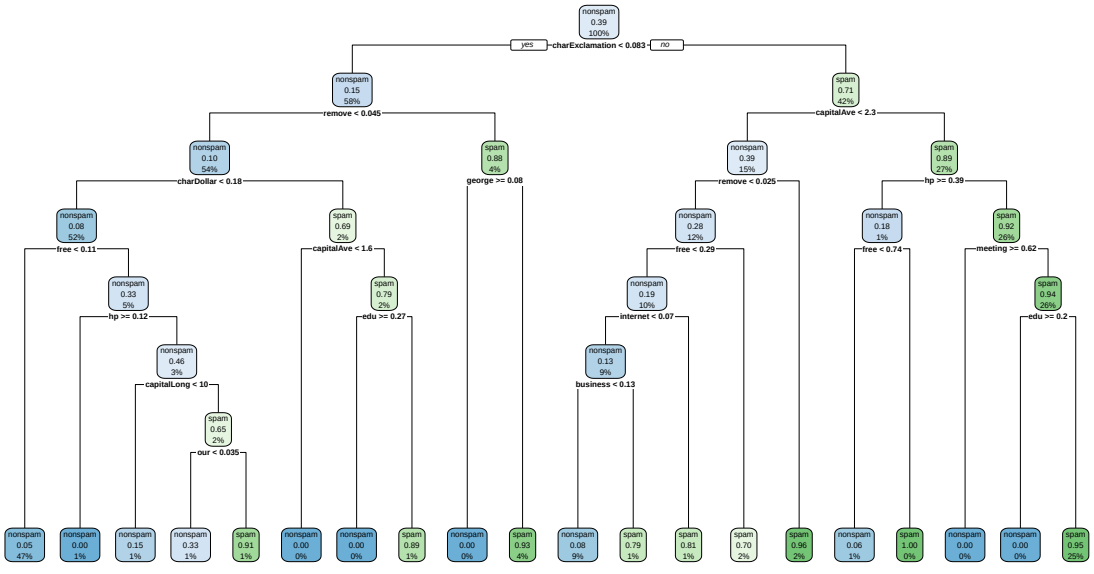
On commence par l'arbre CART

```r
set.seed(123)
arbre <- rpart(type~.,data=dapp,cp=0.0001,minsplit=15)
plotcp(arbre)
```



Les erreurs de prévision ont l'air de se stabiliser à partir de cp=0.0035 :

```
arbre_final <- prune(arbre,cp=0.0035)
rpart.plot(arbre_final)
```



On calcule les prévisions

```
prev_arbre <- predict(arbre_final,newdata=dtest)[,2]
tbl.prev$Arbre <- prev_arbre
```

On passe à la forêt aléatoire.

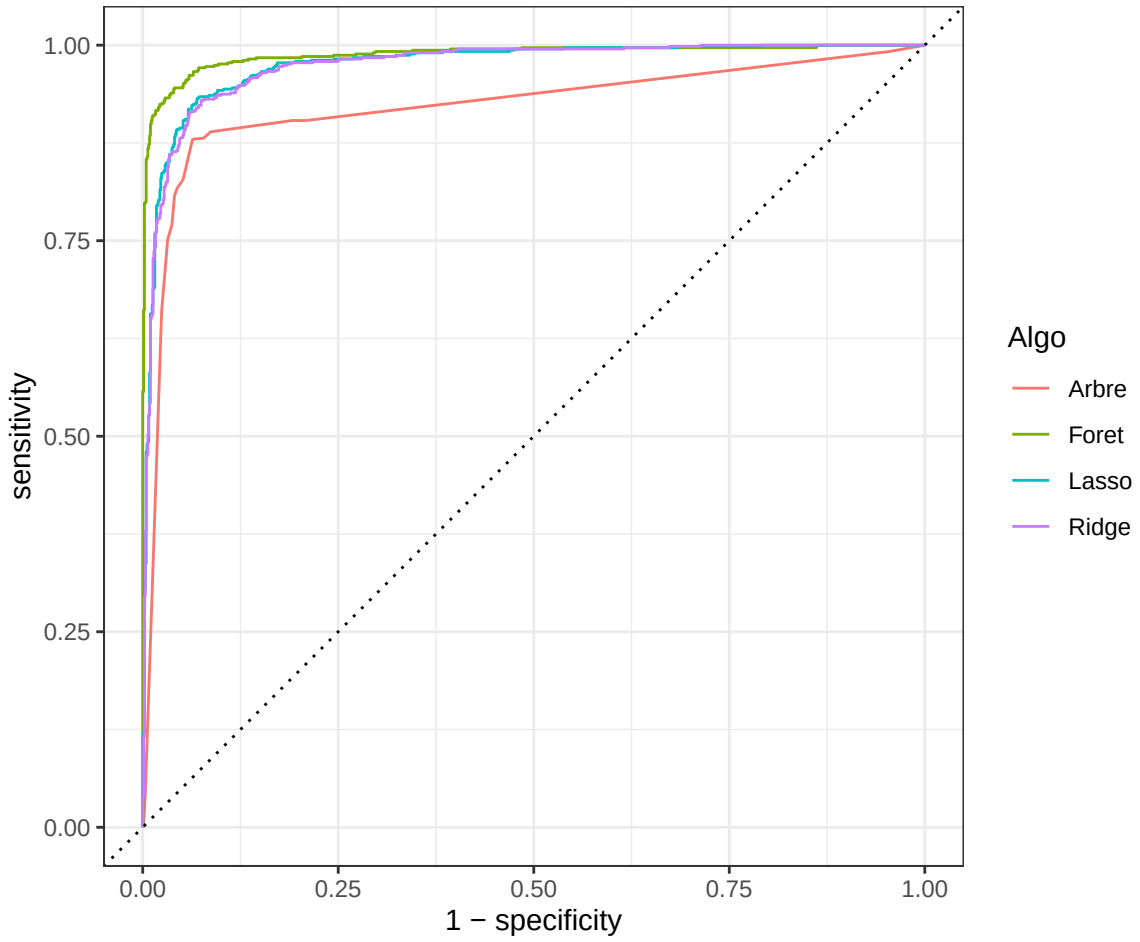
```
foret.prob <- ranger(type~.,data=dapp,probability=TRUE)
```

Les prévisions sur les données `dtest` s'obtiennent avec

```
prev_foret <- predict(foret.prob,data=dtest)$predictions[,2]
tbl.prev$Foret <- prev_foret
```

On peut maintenant tracer les courbes ROC :

```
tbl.prev %>% mutate(obs=dtest$type) %>%
  pivot_longer(-obs,names_to="Algo",values_to = "score") %>%
  group_by(Algo) %>%
  roc_curve(truth=obs,estimate=score,event_level="second") %>%
  autoplot()
```



et calculer les AUC

```
tbl.prev %>% mutate(obs=dtest$type) %>%
  summarize_at(1:4,~roc_auc_vec(truth=obs,estimate=.,event_level="second"))
## # A tibble: 1 x 4
##   Ridge Lasso Arbre Foret
```

```
##   <dbl> <dbl> <dbl> <dbl>
## 1 0.974 0.975 0.921 0.988
```

De nombreux critères comme l'**accuracy**, le **F1-score**, le **kappa de Cohen** sont basés sur la prévision des classes. Cette prévision s'obtient en comparant la probabilité estimée $\mathbf{P}(Y = spam|X = x)$ à un seuil $s \in [0, 1]$. Par exemple avec le seuil 0.5 :

```
prev.class <- round(tbl.prev) %>%
  mutate_all(~dplyr::recode(.,"0"="nonspam","1"="spam")) %>%
  bind_cols(obs=dtest$type)
head(prev.class)
## # A tibble: 6 x 5
##   Ridge Lasso Arbre Foret obs
##   <chr> <chr> <chr> <chr> <fct>
## 1 spam  spam  spam  spam  spam
## 2 spam  spam  spam  spam  spam
## 3 spam  spam  spam  spam  spam
## 4 spam  spam  spam  spam  spam
## 5 spam  spam  spam  spam  spam
## 6 spam  spam  spam  spam  spam
```

Les valeurs des différents critères peuvent s'obtenir à l'aide des fonctions du package `yardstick` :

```
multi_metric <- metric_set(accuracy,bal_accuracy,f_meas,kap)
prev.class %>%
  pivot_longer(-obs,names_to = "Algo",values_to = "classe") %>%
  mutate(classe=as.factor(classe)) %>%
  group_by(Algo) %>%
  multi_metric(truth=obs,estimate=classe,event_level = "second") %>%
  mutate(.estimate=round(.estimate,3)) %>%
  pivot_wider(-.estimator,names_from=.metric,values_from = .estimate)
## # A tibble: 4 x 5
##   Algo  accuracy bal_accuracy f_meas   kap
##   <chr>    <dbl>        <dbl>  <dbl> <dbl>
## 1 Arbre    0.913        0.908  0.892 0.819
## 2 Foret    0.954        0.949  0.942 0.903
## 3 Lasso    0.924        0.916  0.903 0.841
## 4 Ridge    0.92         0.913  0.899 0.833
```

Les forêts aléatoires sont toujours en tête.

Les méthodes ont été comparées par une procédure de validation hold out. Elle présente l'avantage d'être simple mais l'inconvénient de manquer de précision au niveau de l'estimation des critères. Il est en effet préférable d'utiliser des validations croisées, voire même de les répéter. On pourra consulter https://lrouviere.github.io/TUTO_ML/correction/comp-algo.html où une validation croisée est effectuée pour estimer les critères.