

Projet 3/5 Classification Supervisée

Nicolas SALVAN - Alexandre CORRIOU

2024-05-23

Ce document contient le code pour *modéliser les données*. Nous allons réaliser de l'apprentissage supervisé pour prédire le diagnostic des patientes.

Lecture des données nettoyées

Importation du dataset

```
data <- read.csv("data/data_cleaned.csv", header = TRUE, sep = ",")
data$diagnosis <- as.factor(data$diagnosis)

train_data <- read.csv("data/train_data.csv", header = TRUE, sep = ",")
train_data$diagnosis <- as.factor(train_data$diagnosis)

test_data <- read.csv("data/test_data.csv", header = TRUE, sep = ",")
test_data$diagnosis <- as.factor(test_data$diagnosis)
```

Aperçu rapide

```
head(data)
```

```
##      diagnosis radius_mean texture_mean perimeter_mean area_mean smoothness_mean
## 1           M      17.99       10.38         122.80      1001.0         0.11840
## 2           M      20.57       17.77         132.90      1326.0         0.08474
## 3           M      19.69       21.25         130.00      1203.0         0.10960
## 4           M      11.42       20.38          77.58       386.1         0.14250
## 5           M      20.29       14.34         135.10      1297.0         0.10030
## 6           M      12.45       15.70          82.57       477.1         0.12780
## compactness_mean concavity_mean concave.points_mean symmetry_mean
## 1          0.27760          0.3001          0.14710          0.2419
## 2          0.07864          0.0869          0.07017          0.1812
## 3          0.15990          0.1974          0.12790          0.2069
## 4          0.28390          0.2414          0.10520          0.2597
## 5          0.13280          0.1980          0.10430          0.1809
## 6          0.17000          0.1578          0.08089          0.2087
## fractal_dimension_mean radius_se texture_se perimeter_se area_se
## 1          0.07871      1.0950      0.9053          8.589 153.40
```

```
## 2          0.05667    0.5435    0.7339          3.398    74.08
## 3          0.05999    0.7456    0.7869          4.585    94.03
## 4          0.09744    0.4956    1.1560          3.445    27.23
## 5          0.05883    0.7572    0.7813          5.438    94.44
## 6          0.07613    0.3345    0.8902          2.217    27.19
## smoothness_se compactness_se concavity_se concave.points_se symmetry_se
## 1          0.006399    0.04904    0.05373          0.01587    0.03003
## 2          0.005225    0.01308    0.01860          0.01340    0.01389
## 3          0.006150    0.04006    0.03832          0.02058    0.02250
## 4          0.009110    0.07458    0.05661          0.01867    0.05963
## 5          0.011490    0.02461    0.05688          0.01885    0.01756
## 6          0.007510    0.03345    0.03672          0.01137    0.02165
## fractal_dimension_se radius_worst texture_worst perimeter_worst area_worst
## 1          0.006193    25.38      17.33      184.60     2019.0
## 2          0.003532    24.99      23.41      158.80     1956.0
## 3          0.004571    23.57      25.53      152.50     1709.0
## 4          0.009208    14.91      26.50       98.87      567.7
## 5          0.005115    22.54      16.67      152.20     1575.0
## 6          0.005082    15.47      23.75      103.40      741.6
## smoothness_worst compactness_worst concavity_worst concave.points_worst
## 1          0.1622      0.6656      0.7119          0.2654
## 2          0.1238      0.1866      0.2416          0.1860
## 3          0.1444      0.4245      0.4504          0.2430
## 4          0.2098      0.8663      0.6869          0.2575
## 5          0.1374      0.2050      0.4000          0.1625
## 6          0.1791      0.5249      0.5355          0.1741
## symmetry_worst fractal_dimension_worst
## 1          0.4601          0.11890
## 2          0.2750          0.08902
## 3          0.3613          0.08758
## 4          0.6638          0.17300
## 5          0.2364          0.07678
## 6          0.3985          0.12440
```

```
# dim(data)
# str(data)
```

Modélisation

AFD (Analyse Factorielle Discriminante)

Nous allons réaliser une AFD pour prédire le diagnostic des patientes.

```
# install.packages("MASS")
library(MASS)
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attachement du package : 'pROC'
```

```
## Les objets suivants sont masqués depuis 'package:stats':  
##  
##      cov, smooth, var
```

Lancement du modèle

```
afd_lda_model <- lda(diagnosis ~ ., data = train_data)
```

Prédiction

On prédit les données avec le modèle, en précisant les probabilités a priori. On obtient alors la table de confusion suivante.

```
pred_afd <- predict(afd_lda_model, test_data, prior=c(0.5, 0.5))  
table(pred_afd$class, test_data$diagnosis)
```

```
##  
##      B  M  
## B 81  9  
## M  0 24
```

On observe que le modèle a prédit 0 faux négatifs et 9 faux positifs. Peut-être que les données d'entraînement ne sont pas assez représentatives, et que l'on a des tailles de classes différentes.

```
table(train_data$diagnosis)
```

```
##  
##      B  M  
## 276 179
```

```
table(test_data$diagnosis)
```

```
##  
##      B  M  
## 81 33
```

On est en effet sur du 2/3 vs 1/3. Il faudrait rééquilibrer les données pour obtenir des résultats plus fiables.

Performance du modèle

```
afd_accuracy <- mean(pred_afd$class == test_data$diagnosis) # accuracy  
afd_accuracy
```

```
## [1] 0.9210526
```

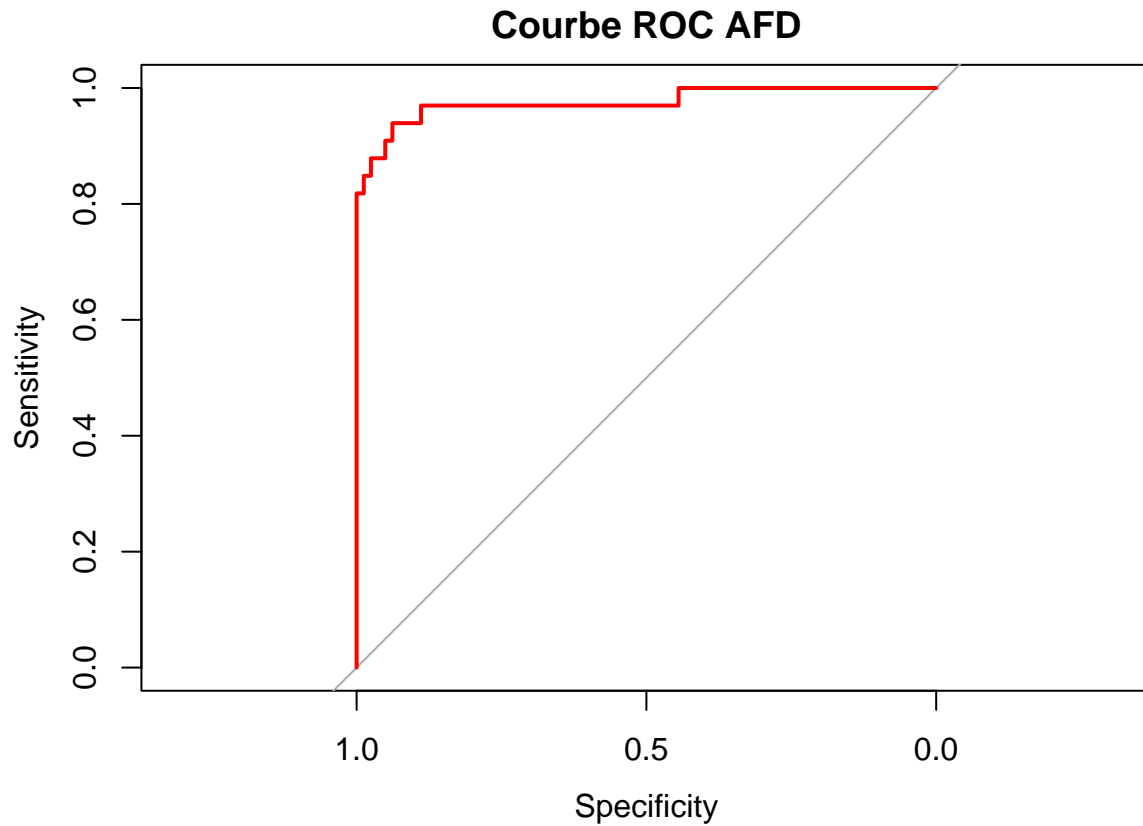
On obtient une accuracy de 0.92, ce qui est plutôt bon.

```
roc_afd <- roc(test_data$diagnosis, pred_afd$posterior[,2])
```

```
## Setting levels: control = B, case = M
```

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

```
plot(roc_afd, col = "red", AUC = TRUE, main = "Courbe ROC AFD")
```



```
auc_afd <- auc(roc_afd)
auc_afd
```

```
## Area under the curve: 0.9753
```

LDA (Analyse Discriminante Linéaire)

Lancement du modèle

```
# Il a déjà été lancé dans la partie AFD
# afd_lda_model <- lda(diagnosis ~ ., data = train_data)
```

Prédiction

```
pred_lda <- predict(afd_lda_model, test_data)
table(pred_lda$class, test_data$diagnosis)
```

```
##
##      B  M
##  B 81 11
##  M   0 22
```

Performance du modèle

```
lda_accuracy <- mean(pred_lda$class == test_data$diagnosis) # accuracy
lda_accuracy
```

```
## [1] 0.9035088
```

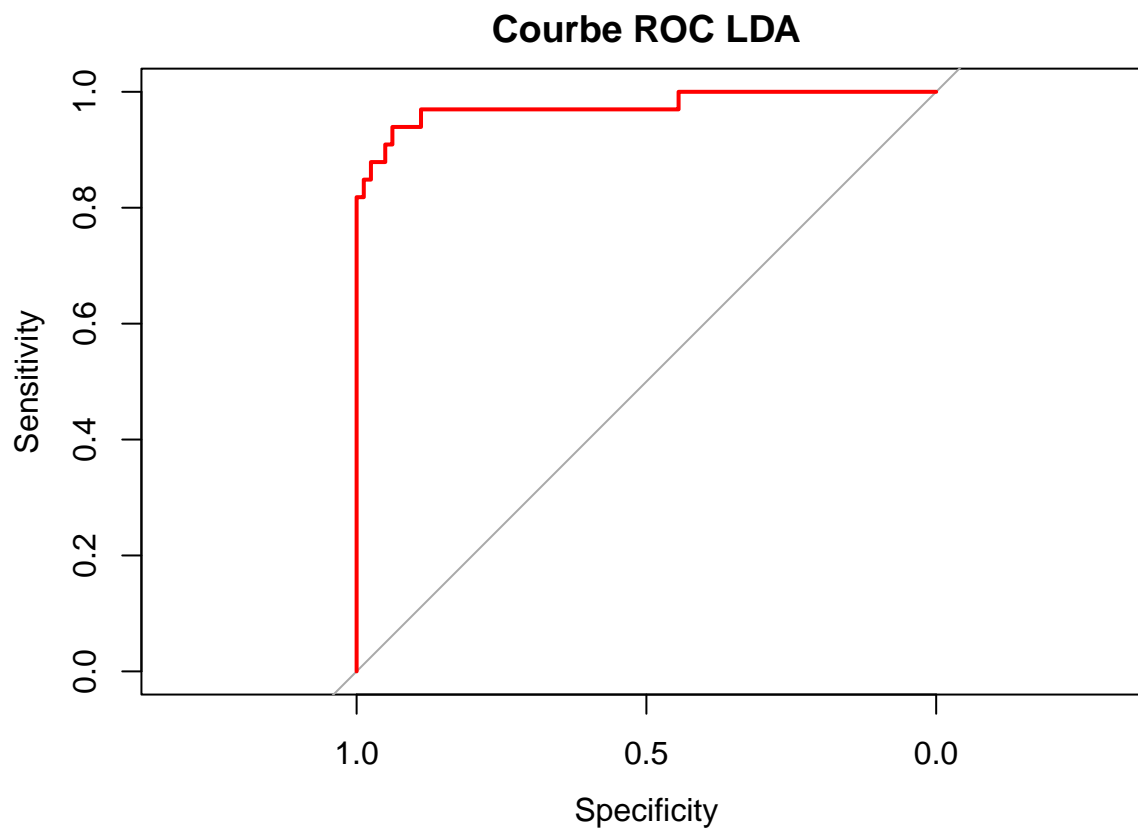
On obtient une accuracy un peu plus faible que l'AFD. Cela s'explique par le fait que l'AFD est plus adaptée aux données.

```
roc_lda <- roc(test_data$diagnosis, pred_lda$posterior[,2])
```

```
## Setting levels: control = B, case = M
```

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

```
plot(roc_lda, col = "red", AUC = TRUE, main = "Courbe ROC LDA")
```



```
auc_lda <- auc(roc_lda)
auc_lda
```

```
## Area under the curve: 0.9753
```

Simplification du modèle

On peut simplifier le modèle en ne prenant que les variables les plus importantes.

```
library(klaR)
```

```
stepwise_lda <- stepclass(diagnosis~., data=train_data, method="lda", direction="backward", output = FA)
```

```
summary(stepwise_lda$model$name)
```

```
##      Length      Class      Mode
##         28      AsIs character
```

On a pu supprimer cinq variables inutiles. On relance une lda sur le modèle simplifié.

```
lda_simple_model <- lda(stepwise_lda$formula, data=train_data)
```

```
pred_lda_simple <- predict(lda_simple_model, test_data)
table(pred_lda_simple$class, test_data$diagnosis)
```

```
##
##      B  M
##  B 81 10
##  M  0 23
```

```
lda_simple_accuracy <- mean(pred_lda_simple$class == test_data$diagnosis) # accuracy
lda_simple_accuracy
```

```
## [1] 0.9122807
```

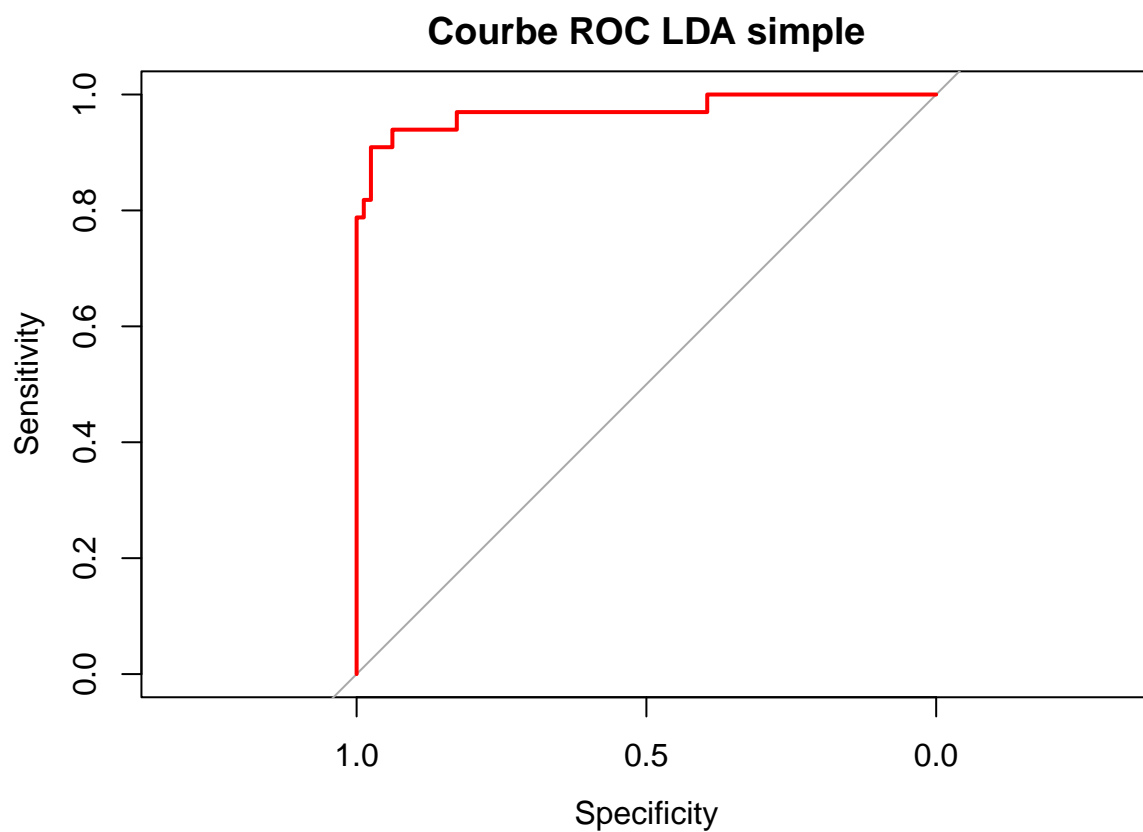
On obtient un score meilleur avec le modèle plus léger. Nous avons peut être fait du sur-apprentissage.

```
roc_lda_simple <- roc(test_data$diagnosis, pred_lda_simple$posterior[,2])
```

```
## Setting levels: control = B, case = M
```

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

```
plot(roc_lda_simple, col = "red", AUC = TRUE, main = "Courbe ROC LDA simple")
```



```
auc_lda_simple <- auc(roc_lda_simple)
auc_lda_simple
```

```
## Area under the curve: 0.9719
```

QDA (Analyse Discriminante Quadratique)

Lancement du modèle

```
afd_qda_model <- qda(diagnosis ~ ., data = train_data)
```

Prédiction

```
pred_qda <- predict(afd_qda_model, test_data)
table(pred_qda$class, test_data$diagnosis)
```

```
##
##      B  M
## B 79  4
## M  2 29
```

Performance du modèle

```
qda_accuracy <- mean(pred_qda$class == test_data$diagnosis) # accuracy
qda_accuracy
```

```
## [1] 0.9473684
```

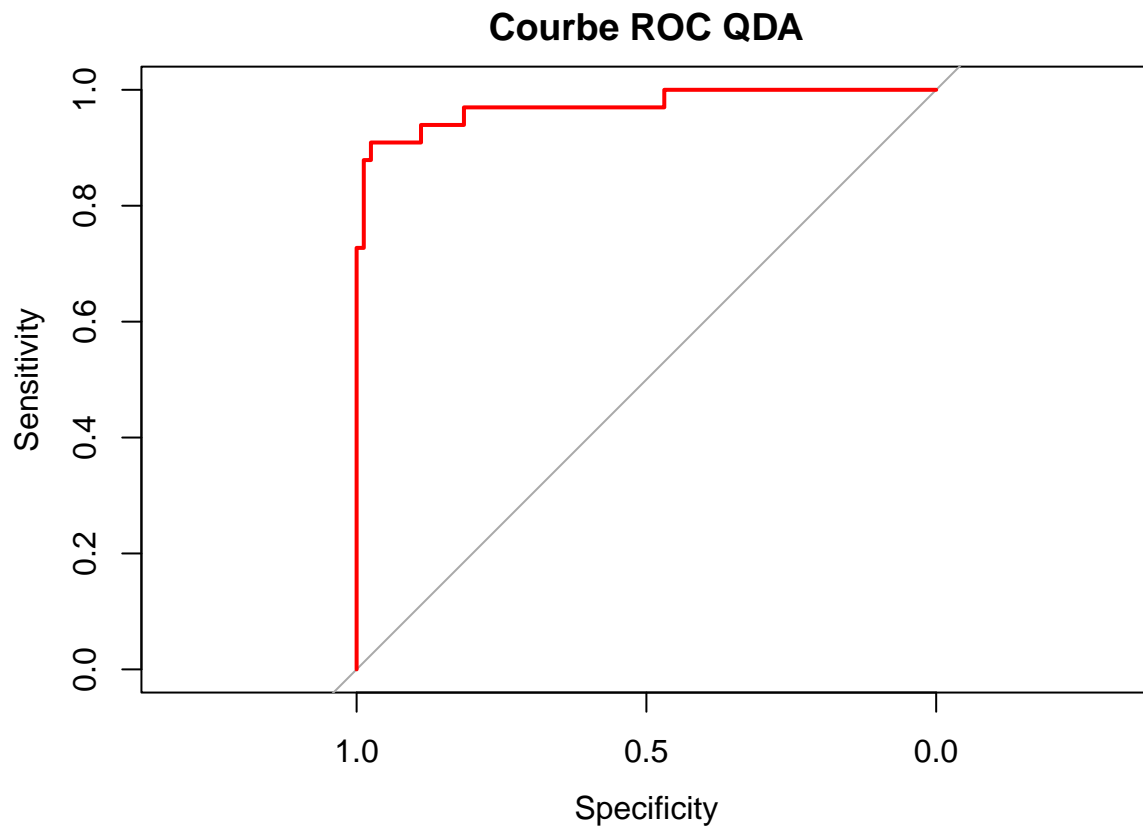
On obtient une accuracy de 0.947, ce qui est plutôt bon.

```
roc_qda <- roc(test_data$diagnosis, pred_qda$posterior[,2])
```

```
## Setting levels: control = B, case = M
```

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

```
plot(roc_qda, col = "red", AUC = TRUE, main = "Courbe ROC QDA")
```

```
auc_qda <- auc(roc_qda)
auc_qda
```

```
## Area under the curve: 0.9723
```

Simplification du modèle

On peut simplifier le modèle en ne prenant que les variables les plus importantes.

```
stepwise_qda <- stepclass(diagnosis~., data=train_data, method="qda", direction="backward", output = FALSE)
```

```
qda_simple_model <- qda(stepwise_qda$formula, data=train_data)
```

```
summary(stepwise_qda$model$name)
```

```
##      Length      Class      Mode
##         27         AsIs character
```

On a pu supprimer deux variables inutiles. On relance une qda sur le modèle simplifié.

```
pred_qda_simple <- predict(qda_simple_model, test_data)
table(pred_qda_simple$class, test_data$diagnosis)
```

```
##
##      B  M
##    B 79  3
##    M  2 30
```

```
qda_simple_accuracy <- mean(pred_qda_simple$class == test_data$diagnosis) # accuracy
qda_simple_accuracy
```

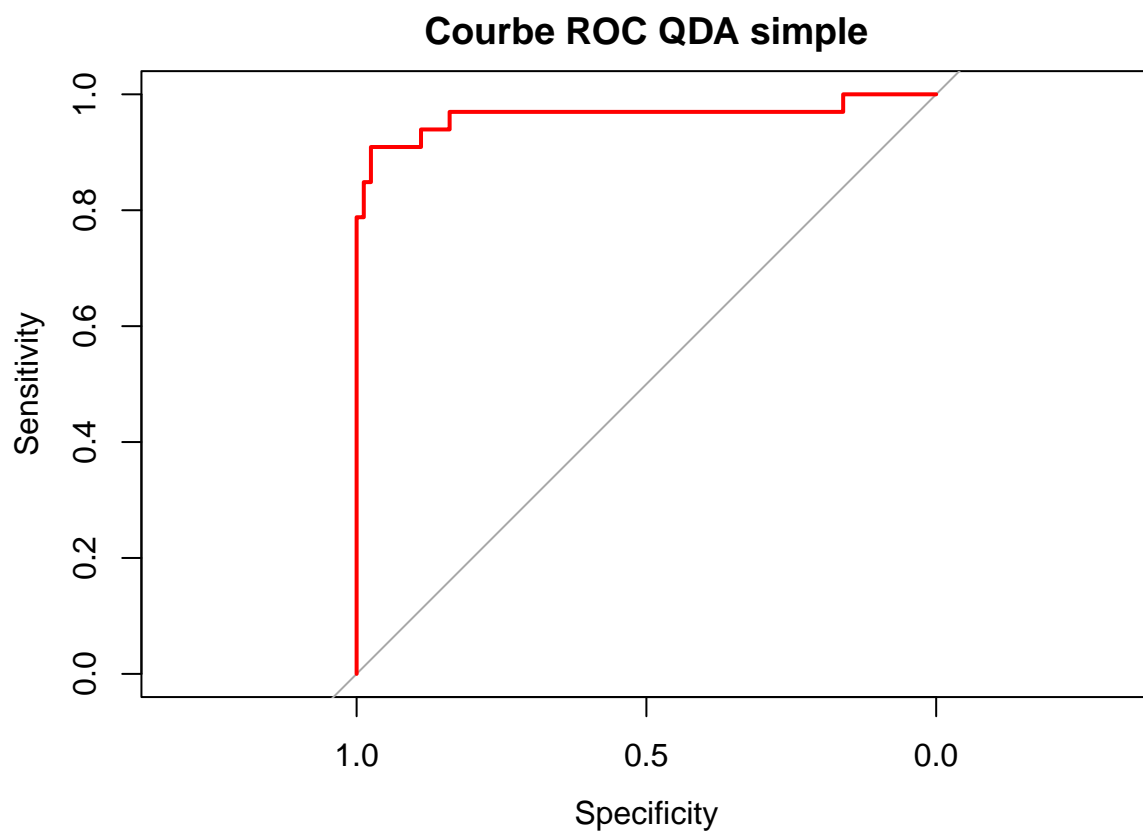
```
## [1] 0.9561404
```

```
roc_qda_simple <- roc(test_data$diagnosis, pred_qda_simple$posterior[,2])
```

```
## Setting levels: control = B, case = M
```

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

```
plot(roc_qda_simple, col = "red", AUC = TRUE, main = "Courbe ROC QDA simple")
```



```
auc_qda_simple <- auc(roc_qda_simple)
auc_qda_simple
```

```
## Area under the curve: 0.9641
```

CART

```
library(rpart.plot)
```

```
## Le chargement a nécessité le package : rpart
```

```
library(rpart)
```

Lancement du modèle

Nous allons construire un arbre naïf, avec les paramètres par défaut.

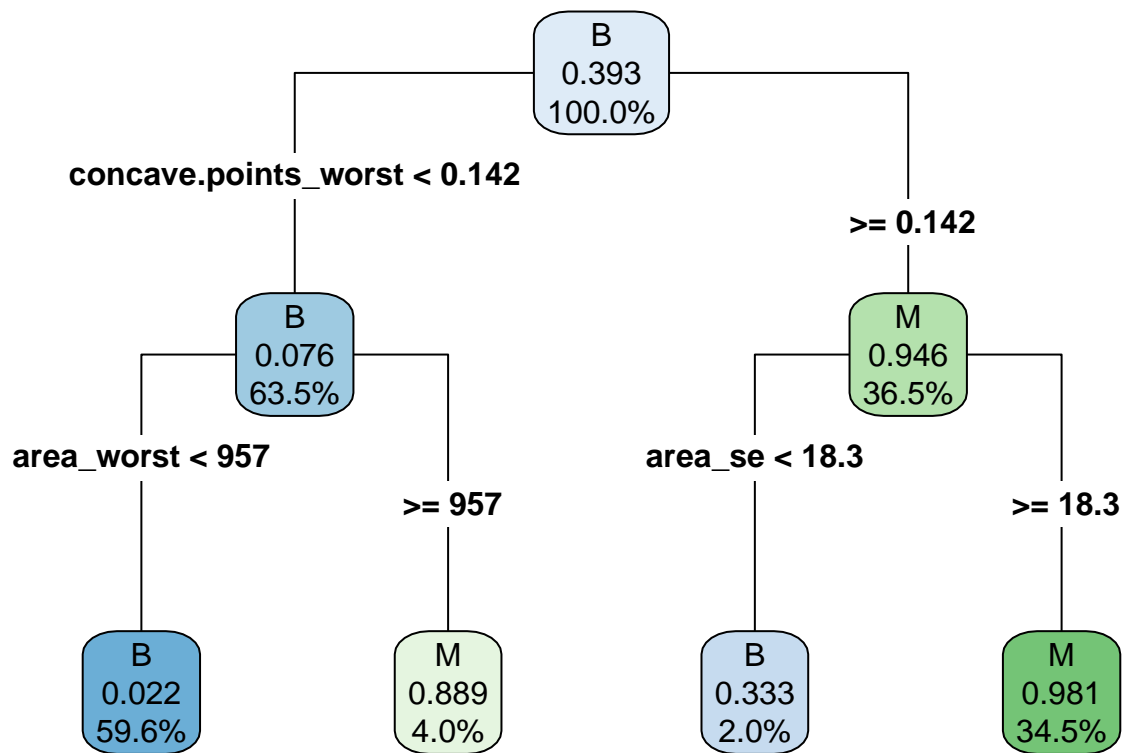
```
tree <- rpart(diagnosis~., train_data)
```

```
print(tree)
```

```
## n= 455
##
## node), split, n, loss, yval, (yprob)
##      * denotes terminal node
##
## 1) root 455 179 B (0.60659341 0.39340659)
##    2) concave.points_worst< 0.14235 289 22 B (0.92387543 0.07612457)
##      4) area_worst< 957.45 271 6 B (0.97785978 0.02214022) *
##      5) area_worst>=957.45 18 2 M (0.11111111 0.88888889) *
##    3) concave.points_worst>=0.14235 166 9 M (0.05421687 0.94578313)
##      6) area_se< 18.335 9 3 B (0.66666667 0.33333333) *
##      7) area_se>=18.335 157 3 M (0.01910828 0.98089172) *
```

Visualisation de l'arbre

```
rpart.plot(tree, type=4, digits=3,roundint=FALSE)
```



Prédiction et performance du modèle naïf

```

pred_cart_naive <- predict(tree, newdata=test_data, type="prob")
pred_cart_naive_qual <- ifelse(pred_cart_naive[,2] > 0.5, "M", "B")
table(pred_cart_naive_qual, test_data$diagnosis)

```

```

##
## pred_cart_naive_qual  B  M
##                      B 78  7
##                      M  3 26

```

```

cart_naive_accuracy <- mean(pred_cart_naive_qual == test_data$diagnosis) # accuracy
cart_naive_accuracy

```

```

## [1] 0.9122807

```

```

roc_cart_naive <- roc(test_data$diagnosis, pred_cart_naive[,2], plot=TRUE, col = "red", main = "Courbe ROC")

```

```

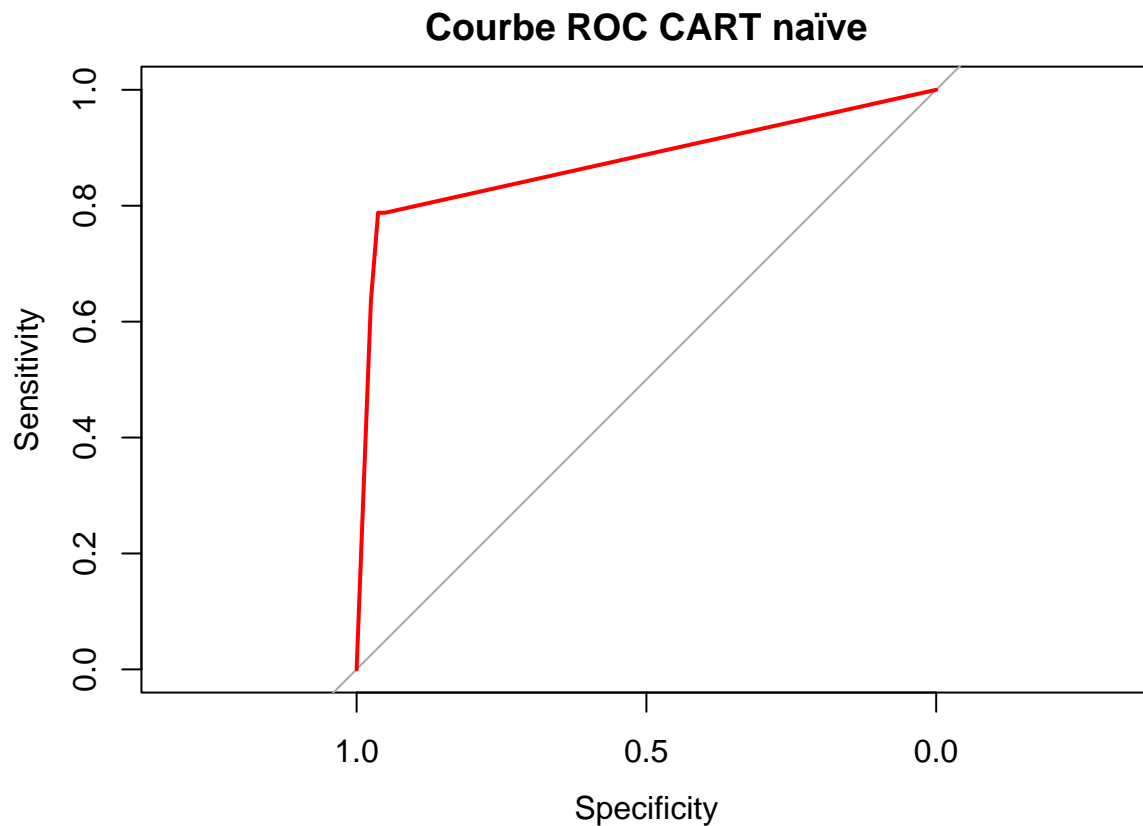
## Setting levels: control = B, case = M

```

```

## Setting direction: controls < cases

```



```
auc_cart_naive <- auc(roc_cart_naive)
auc_cart_naive
```

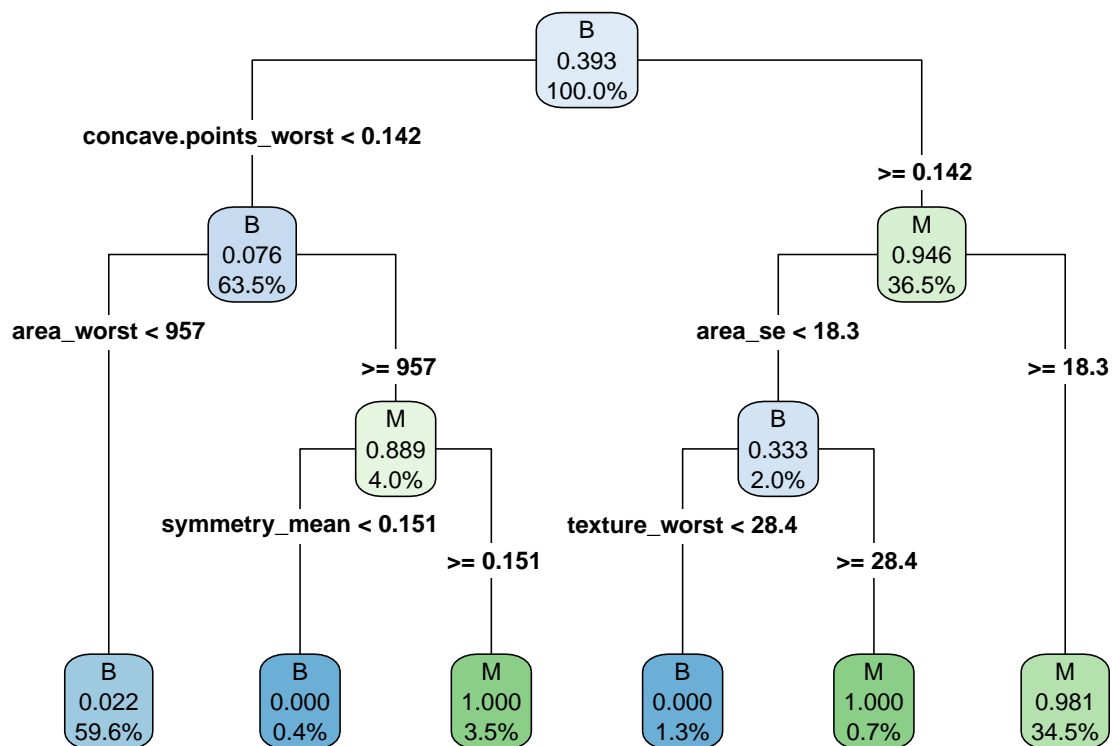
```
## Area under the curve: 0.8762
```

On n'obtient pas de très bonnes performances avec le modèle naïf. Nous allons essayer de l'améliorer.

Arbre simplifié

Nous allons simplifier l'arbre pour éviter le sur-apprentissage.

```
tree2 <- rpart(diagnosis~.,train_data,control=rpart.control(minsplit=5))
rpart.plot(tree2, type=4, digits=3)
```



Prédiction et performance du modèle simplifié

```

pred_cart <- predict(tree2, newdata=test_data, type="prob")
pred_cart_qual <- ifelse(pred_cart[,2] > 0.5, "M", "B")
table(pred_cart_qual, test_data$diagnosis)

```

```

##
## pred_cart_qual  B  M
##                B 78  8
##                M  3 25

```

```

cart_accuracy <- mean(pred_cart_qual == test_data$diagnosis) # accuracy
cart_accuracy

```

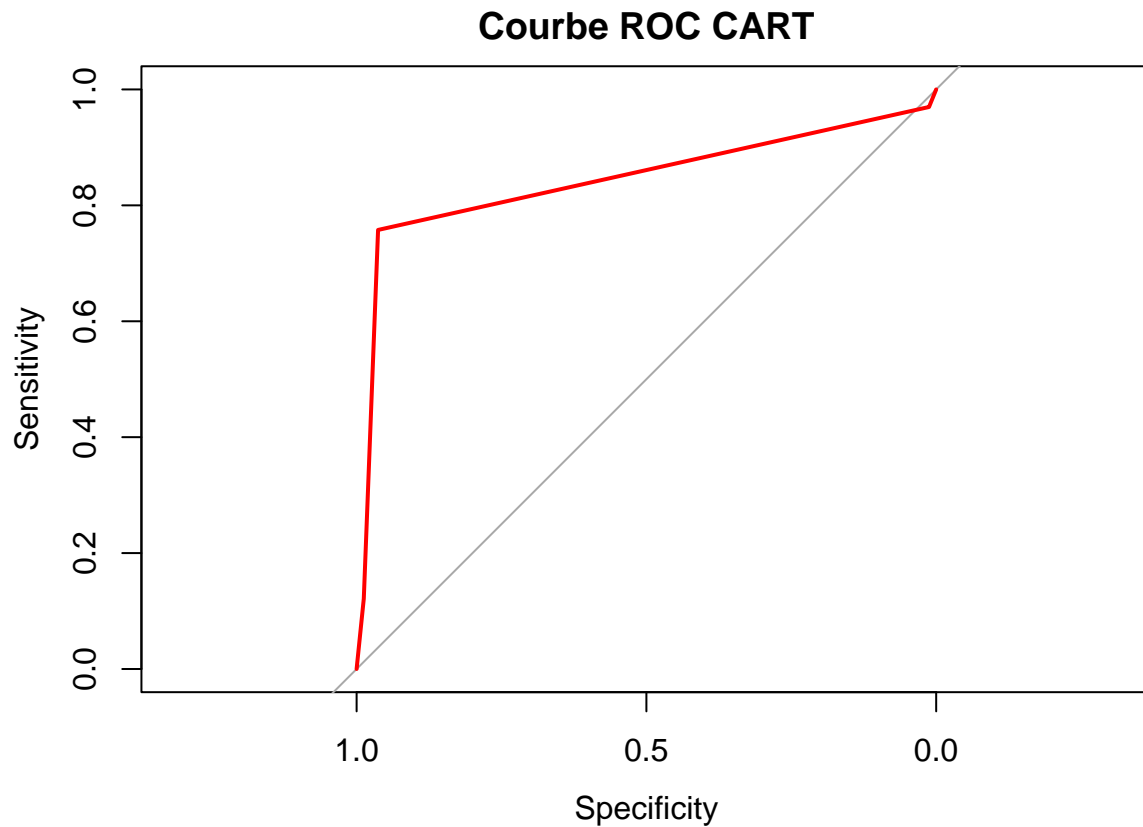
```
## [1] 0.9035088
```

```
roc_cart <- roc(test_data$diagnosis, pred_cart[,2])
```

```
## Setting levels: control = B, case = M
```

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

```
plot(roc_cart, col = "red", AUC = TRUE, main = "Courbe ROC CART")
```



```
auc_cart <- auc(roc_cart)
auc_cart
```

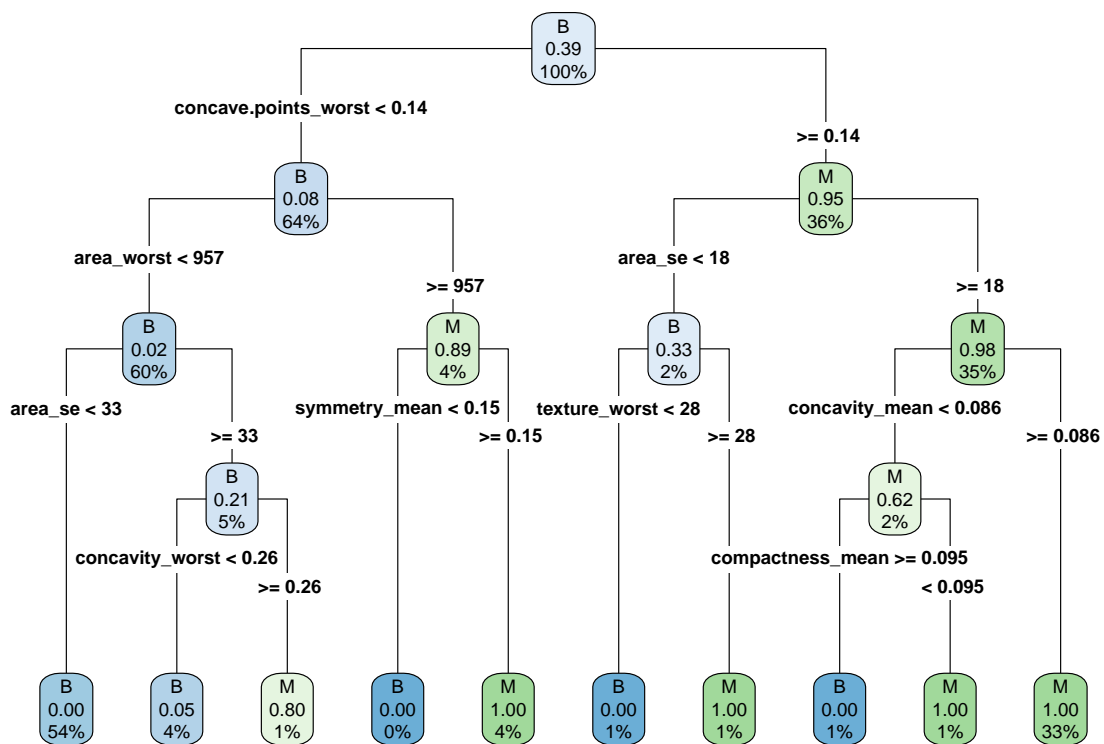
Area under the curve: 0.8447

Cela n'a pas amélioré le modèle. Nous allons essayer de faire de l'élagage.

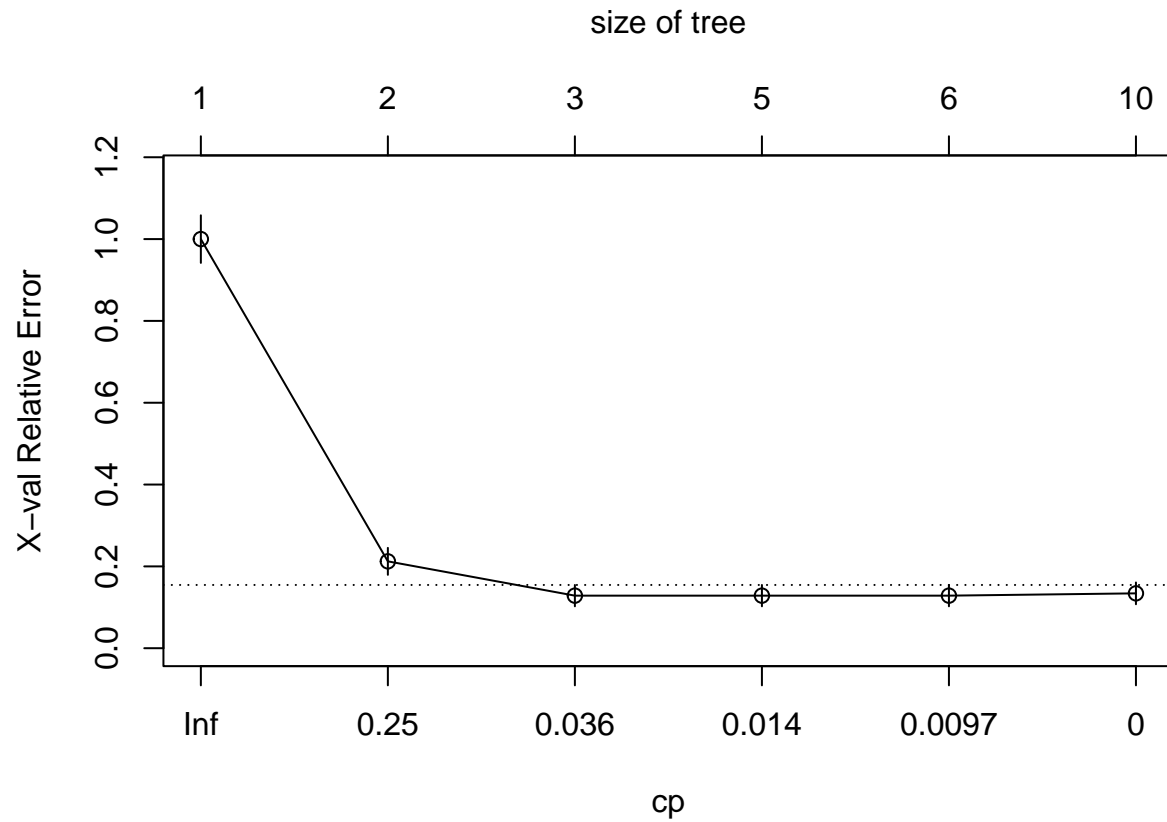
Elagage des arbres

Nous allons élaguer l'arbre pour éviter le sur-apprentissage.

```
tree_elag <- rpart(diagnosis~.,train_data,control=rpart.control(minsplit=5,cp=0))
rpart.plot(tree_elag, type=4)
```

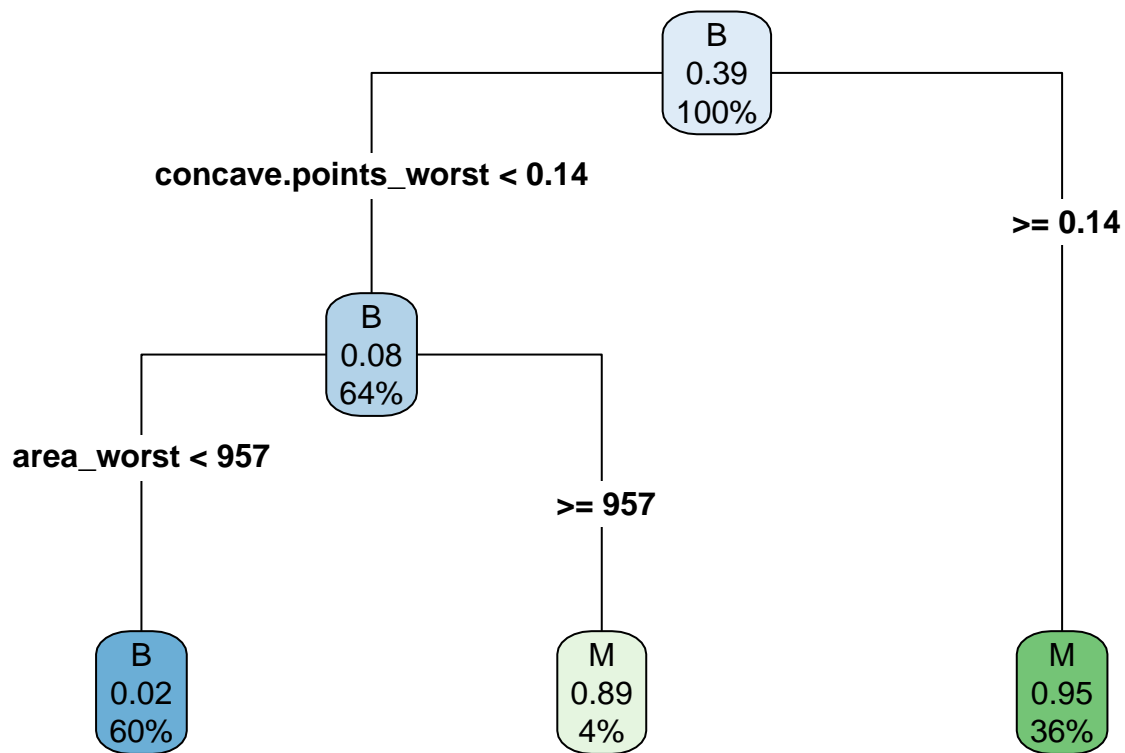


```
plotcp(tree_elag)
```

On observe une erreur minimale pour arbre de taille entre 4 et 7.

```
cp.opt <- tree_elag$cptable[which.min(tree_elag$cptable[, "xerror"]), "CP"]
tree.opt <- prune(tree_elag, cp=cp.opt)
rpart.plot(tree.opt, type=4)
```



Prediction

```

pred_cart_opti <- predict(tree.opt, newdata=test_data, type="prob")
pred_cart_opti_qual <- ifelse(pred_cart_opti[,2] > 0.5, "M", "B")
table(pred_cart_opti_qual, test_data$diagnosis)

```

```

##
## pred_cart_opti_qual  B  M
##                   B 77  7
##                   M  4 26

```

Performance du modèle

```

cart_opti_accuracy <- mean(pred_cart_opti_qual == test_data$diagnosis) # accuracy
cart_opti_accuracy

```

```
## [1] 0.9035088
```

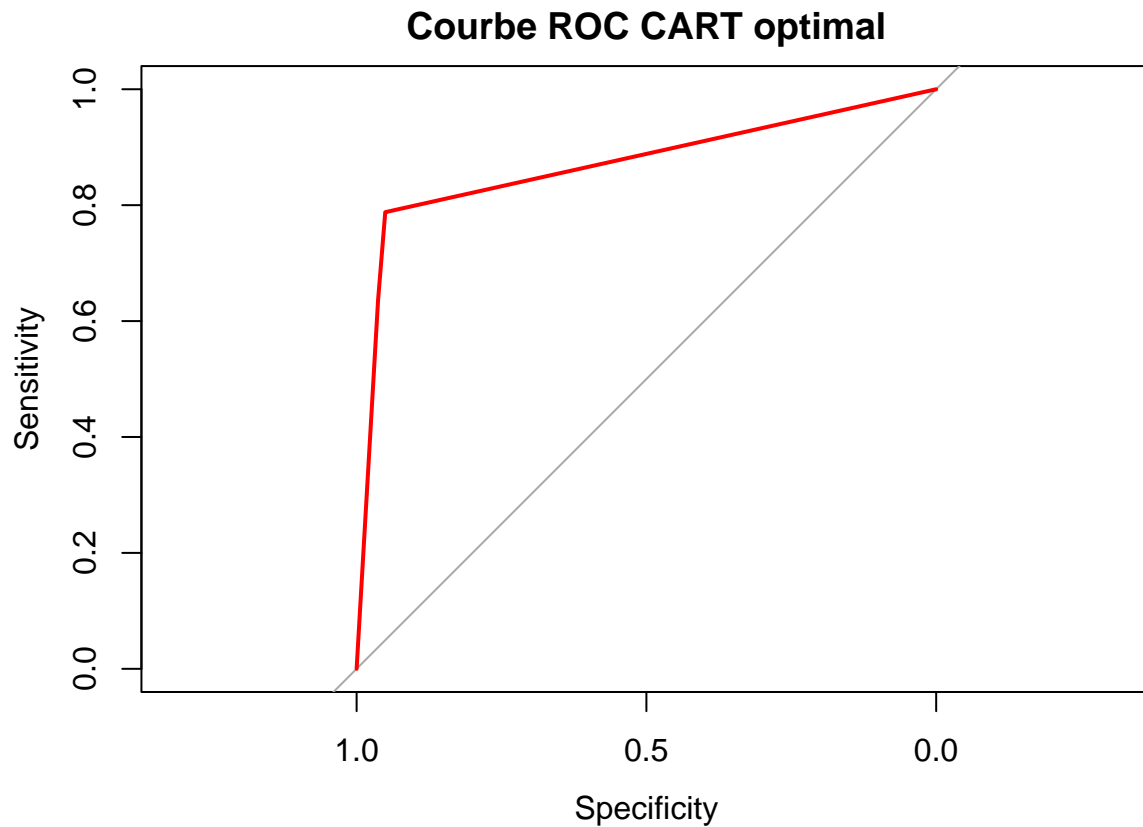
Pour le modèle CART optimal, on obtient une accuracy de 0.91, ce qui est meilleur que les version précédentes de CART.

```
roc_cart_opti <- roc(test_data$diagnosis, pred_cart_opti[,2])
```

```
## Setting levels: control = B, case = M
```

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

```
plot(roc_cart_opti, col = "red", AUC = TRUE, main = "Courbe ROC CART optimal")
```



```
auc_cart_opti <- auc(roc_cart_opti)  
auc_cart_opti
```

```
## Area under the curve: 0.8704
```

On obtient une AUC de 0.8788, ce qui est moins bon que les modèles précédents.

Random Forest

Lancement du modèle

```
# install.packages("randomForest")  
library(randomForest)
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
rf_model <- randomForest(train_data$diagnosis ~ ., data = train_data, ntree = 100)
```

Prédiction

```
pred_rf <- predict(rf_model, test_data, type="prob")
pred_rf_fact <- ifelse(pred_rf[,2] > 0.5, "M", "B")
prob_rf <- pred_rf[, "B"]
table(pred_rf_fact, test_data$diagnosis)
```

```
##
## pred_rf_fact  B  M
##              B 79  6
##              M  2 27
```

Performance du modèle

```
rf_accuracy <- mean(pred_rf_fact == test_data$diagnosis) # accuracy
rf_accuracy
```

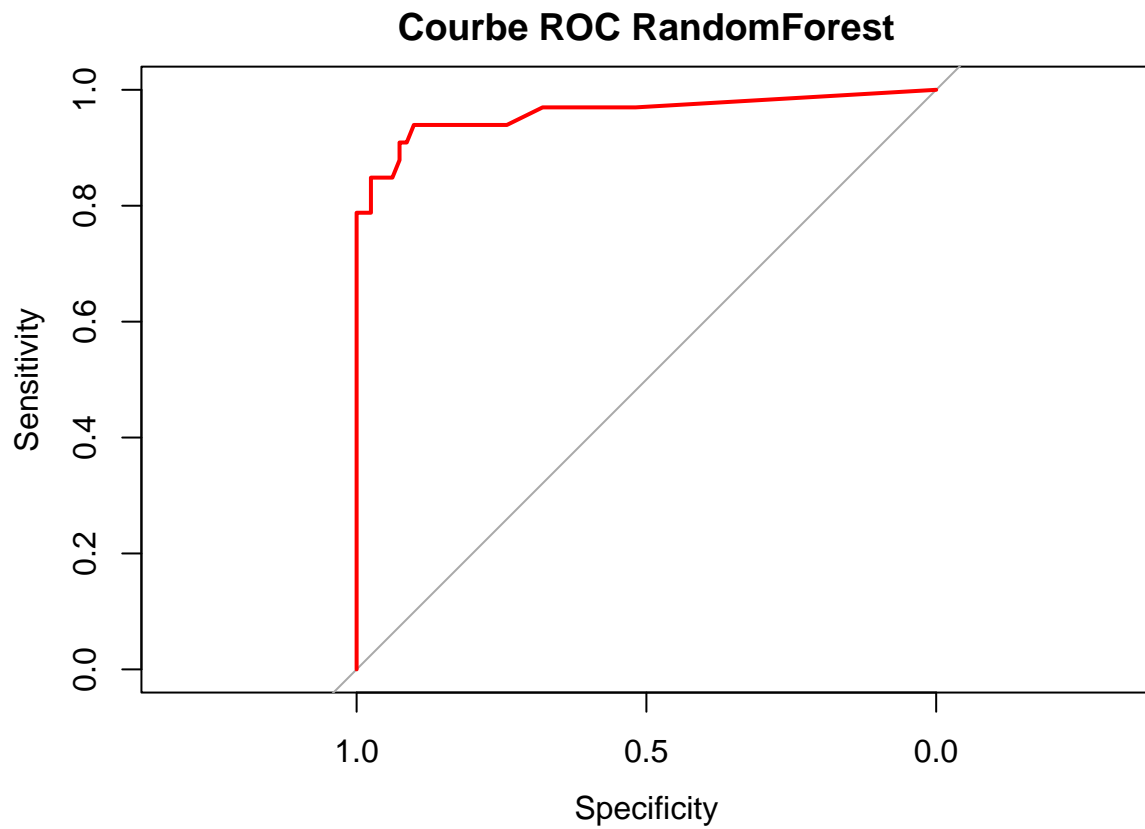
```
## [1] 0.9298246
```

```
roc_rf <- roc(test_data$diagnosis, prob_rf)
```

```
## Setting levels: control = B, case = M
```

```
## Setting direction: controls > cases
```

```
plot(roc_rf, col = "red", AUC = TRUE, main = "Courbe ROC RandomForest")
```

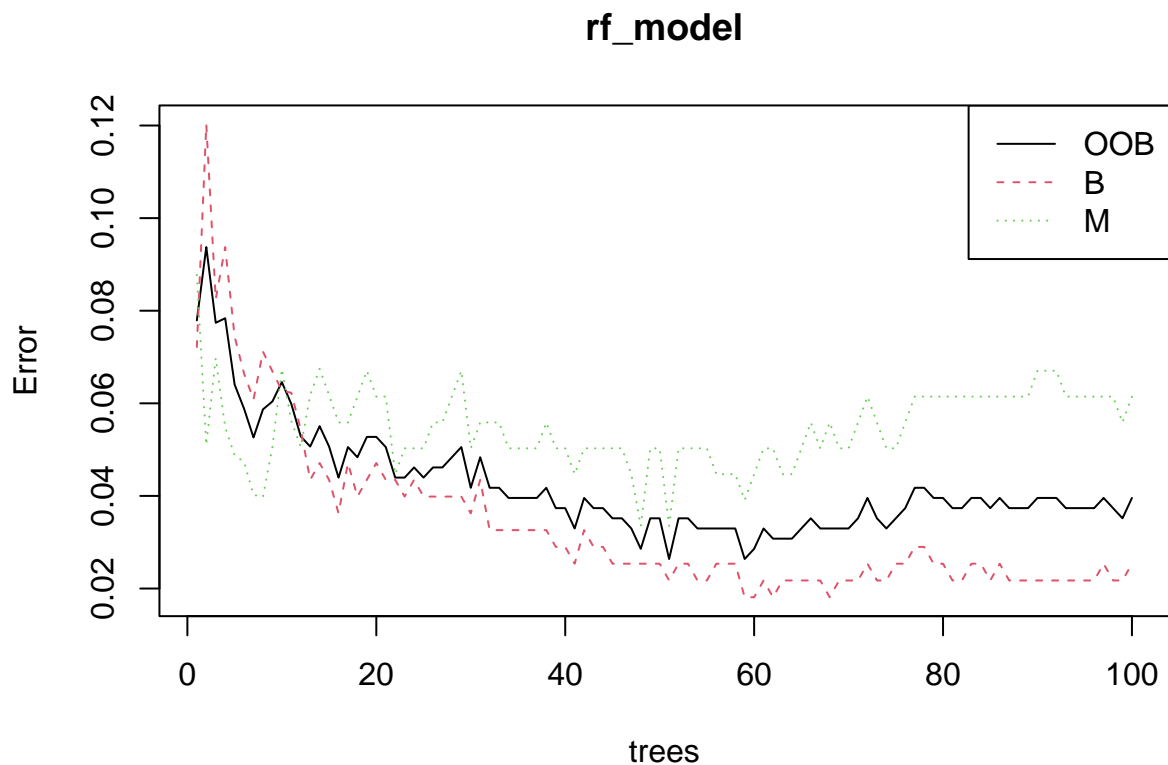


```
auc_rf <- auc(roc_rf)
auc_rf
```

Area under the curve: 0.9602

Erreur OOB (Out-of-Bag)

```
plot(rf_model)
legend("topright", colnames(rf_model$err.rate), col=1:3, lty=1:3)
```



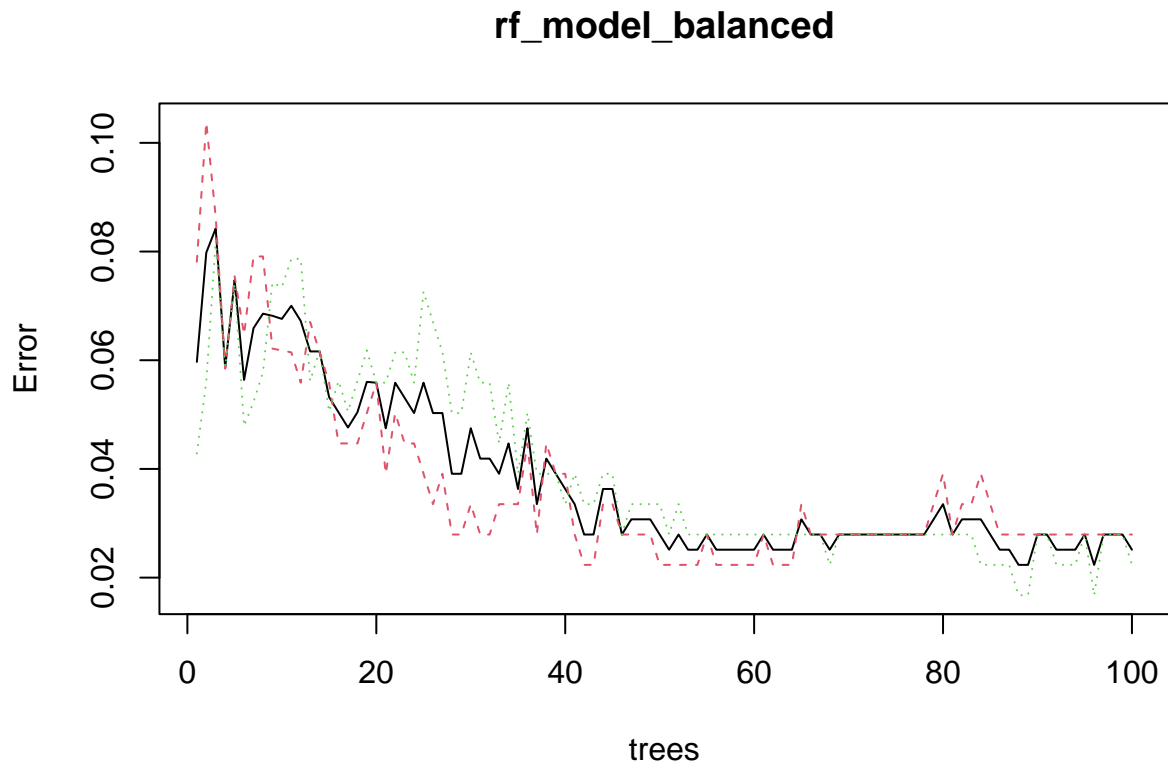
Le taux d'erreur semble s'être stabilisé. On observe que l'erreur OOB pour la classification "M" semble plus élevée que pour la classification "B". Il y a donc un déséquilibre dans les classes. Essayons d'équilibrer le jeu d'entraînement.

Modèle avec données équilibrées

```
train_data_balanced <- read.csv("data/train_data_balanced.csv", header = TRUE, sep = ",")
train_data_balanced$diagnosis <- as.factor(train_data_balanced$diagnosis)
```

```
rf_model_balanced <- randomForest(train_data_balanced$diagnosis ~ ., data = train_data_balanced, ntree = 100)
```

```
plot(rf_model_balanced)
```



Le taux d'erreur semble s'être stabilisé.

```
pred_rf_balanced <- predict(rf_model_balanced, test_data, type="prob")
pred_rf_fact_balanced <- ifelse(pred_rf_balanced[,2] > 0.5, "M", "B")
prob_rf_balanced <- pred_rf_balanced[, "B"]
table(pred_rf_fact_balanced, test_data$diagnosis)
```

```
##
## pred_rf_fact_balanced  B  M
##                      B 78  6
##                      M  3 27
```

```
rf_accuracy_balanced <- mean(pred_rf_fact_balanced == test_data$diagnosis) # accuracy
rf_accuracy_balanced
```

```
## [1] 0.9210526
```

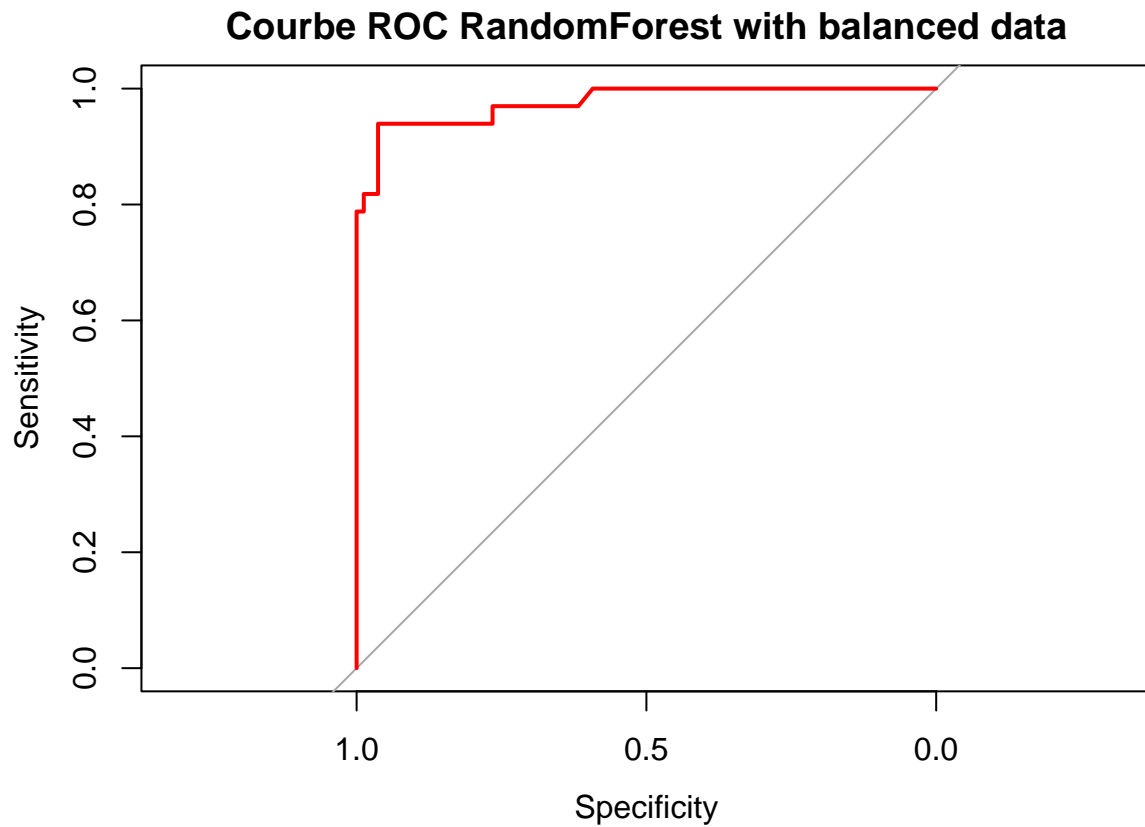
On obtient des performances similaires.

```
roc_rf_balanced <- roc(test_data$diagnosis, prob_rf_balanced)
```

```
## Setting levels: control = B, case = M
```

```
## Setting direction: controls > cases
```

```
plot(roc_rf_balanced, col = "red", AUC = TRUE, main = "Courbe ROC RandomForest with balanced data")
```



```
auc_rf_balanced <- auc(roc_rf_balanced)
auc_rf_balanced
```

```
## Area under the curve: 0.9761
```

Régression Logistique

Lancement du modèle

```
glm_model <- glm(diagnosis ~ ., data = train_data, family = binomial, control = glm.control(maxit = 50))
```

```
## Warning: glm.fit: des probabilités ont été ajustées numériquement à 0 ou 1
```

Il y a des warnings, ce qui signifie que le modèle n'est pas bien calibré.

```
summary(glm_model)
```

```
##
## Call:
```



```

## glm(formula = diagnosis ~ ., family = binomial, data = train_data,
##      control = glm.control(maxit = 50))
##
## Deviance Residuals:
##      Min        1Q      Median        3Q      Max
## -7.485e-06 -2.110e-08 -2.110e-08  2.110e-08  6.376e-06
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -9.329e+02  2.877e+07      0      1
## radius_mean      2.774e+02  4.530e+06      0      1
## texture_mean      7.408e+00  1.584e+05      0      1
## perimeter_mean   -4.363e+01  6.753e+05      0      1
## area_mean        1.673e-01  2.183e+04      0      1
## smoothness_mean  -2.848e+03  7.949e+07      0      1
## compactness_mean -1.647e+03  3.579e+07      0      1
## concavity_mean   -5.379e+02  1.613e+07      0      1
## concave.points_mean 4.974e+03  7.770e+07      0      1
## symmetry_mean    -3.795e+01  4.227e+07      0      1
## fractal_dimension_mean 6.281e+03  9.972e+07      0      1
## radius_se        1.546e+03  1.939e+07      0      1
## texture_se        4.741e+01  2.238e+06      0      1
## perimeter_se     -1.366e+02  1.256e+06      0      1
## area_se          -3.931e+00  1.295e+05      0      1
## smoothness_se    -2.454e+04  2.655e+08      0      1
## compactness_se   -1.251e+03  1.211e+08      0      1
## concavity_se      5.385e+02  6.007e+07      0      1
## concave.points_se  8.235e+03  9.647e+07      0      1
## symmetry_se      -1.094e+03  1.026e+08      0      1
## fractal_dimension_se -4.290e+03  4.748e+08      0      1
## radius_worst     -1.891e+02  8.543e+05      0      1
## texture_worst      1.326e+00  2.978e+05      0      1
## perimeter_worst    1.938e+01  1.137e+05      0      1
## area_worst        9.042e-01  1.133e+04      0      1
## smoothness_worst   3.807e+03  4.315e+07      0      1
## compactness_worst -3.655e+02  2.026e+07      0      1
## concavity_worst    4.967e+02  1.215e+07      0      1
## concave.points_worst -5.145e+02  2.301e+07      0      1
## symmetry_worst      8.065e+02  2.182e+07      0      1
## fractal_dimension_worst 1.398e+03  6.463e+07      0      1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 6.0993e+02  on 454  degrees of freedom
## Residual deviance: 4.0275e-10  on 424  degrees of freedom
## AIC: 62
##
## Number of Fisher Scoring iterations: 31

```

Calcul des odds ratios (OR)

```
exp(coef(glm_model))
```

```
##           (Intercept)           radius_mean           texture_mean
##           0.000000e+00           2.970743e+120           1.649803e+03
##           perimeter_mean           area_mean           smoothness_mean
##           1.127913e-19           1.182149e+00           0.000000e+00
##           compactness_mean           concavity_mean           concave.points_mean
##           0.000000e+00           2.540332e-234           Inf
##           symmetry_mean fractal_dimension_mean           radius_se
##           3.303280e-17           Inf           Inf
##           texture_se           perimeter_se           area_se
##           3.900971e+20           4.791425e-60           1.962475e-02
##           smoothness_se           compactness_se           concavity_se
##           0.000000e+00           0.000000e+00           7.466853e+233
##           concave.points_se           symmetry_se           fractal_dimension_se
##           Inf           0.000000e+00           0.000000e+00
##           radius_worst           texture_worst           perimeter_worst
##           7.667257e-83           3.764820e+00           2.616095e+08
##           area_worst           smoothness_worst           compactness_worst
##           2.469858e+00           Inf           1.929610e-159
##           concavity_worst           concave.points_worst           symmetry_worst
##           5.313389e+215           3.556905e-224           Inf
## fractal_dimension_worst
##           Inf
```

On observe des extrêmes pour les odds ratios. Cela est dû à la présence de variables corrélées, nous allons donc plus tard simplifier le modèle.

Intérêt des variables explicatives

```
res0 =glm(diagnosis ~ 1, family = "binomial", data=train_data)
anova(res0,glm_model,test="Chisq")
```

```
## Analysis of Deviance Table
##
## Model 1: diagnosis ~ 1
## Model 2: diagnosis ~ radius_mean + texture_mean + perimeter_mean + area_mean +
##           smoothness_mean + compactness_mean + concavity_mean + concave.points_mean +
##           symmetry_mean + fractal_dimension_mean + radius_se + texture_se +
##           perimeter_se + area_se + smoothness_se + compactness_se +
##           concavity_se + concave.points_se + symmetry_se + fractal_dimension_se +
##           radius_worst + texture_worst + perimeter_worst + area_worst +
##           smoothness_worst + compactness_worst + concavity_worst +
##           concave.points_worst + symmetry_worst + fractal_dimension_worst
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1         454         609.93
## 2         424           0.00 30    609.93 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

On observe que l'on a une p-value proche de 0, donc on rejette l'hypothèse nulle et on peut conclure qu'il y a au moins une variables explicative qui est significative.

Prédiction

```
pred_glm <- predict(glm_model, test_data, type = "response")
pred_glm_qual <- ifelse(pred_glm > 0.5, "M", "B")
table(pred_glm_qual, test_data$diagnosis)
```

```
##
## pred_glm_qual  B  M
##               B 75  7
##               M  6 26
```

Performance du modèle

```
glm_accuracy <- mean(pred_glm_qual == test_data$diagnosis) # accuracy
glm_accuracy
```

```
## [1] 0.8859649
```

On obtient une accuracy de 88.6%, ce qui est plutôt bon mais pas aussi bon que les autres modèles.

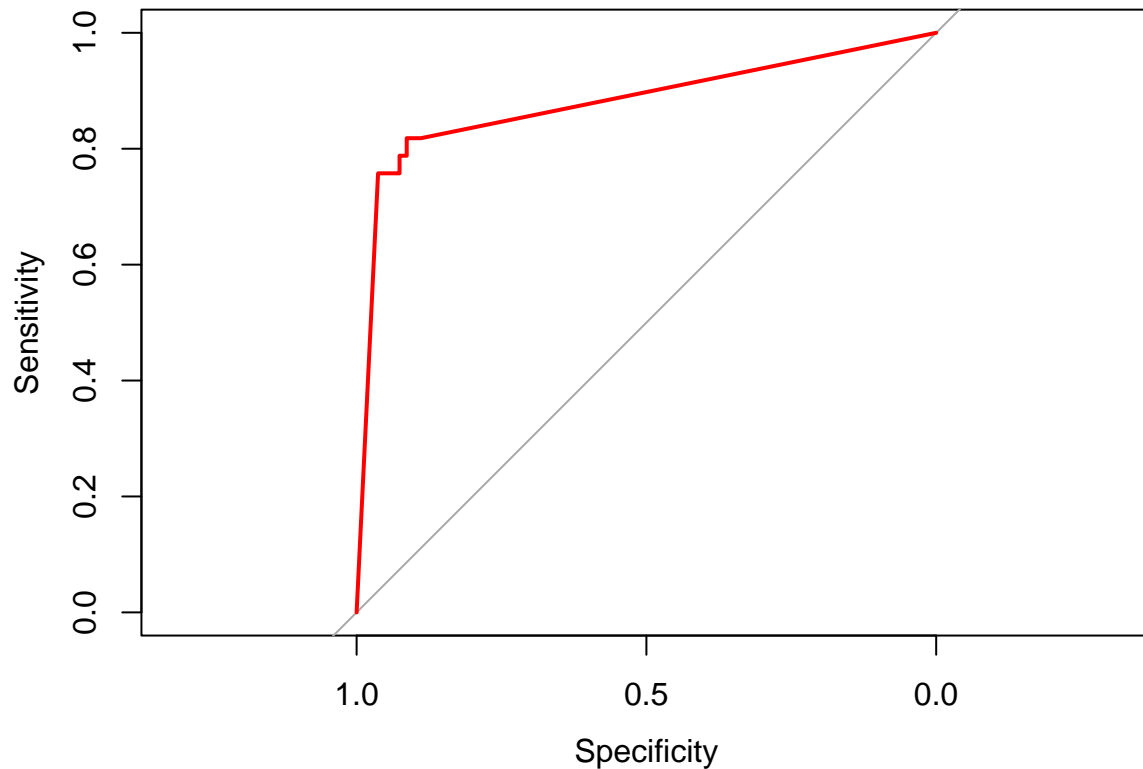
```
roc_glm <- roc(test_data$diagnosis, pred_glm)
```

```
## Setting levels: control = B, case = M
```

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

```
plot(roc_glm, col = "red", AUC = TRUE, main = "Courbe ROC Régression logistique multiple")
```

Courbe ROC Régression logstique multiple



```
auc_glm <- auc(roc_glm)
auc_glm
```

```
## Area under the curve: 0.8801
```

Simplification du modèle avec des régressions logistiques pénalisées

Nous allons simplifier le modèle de régression logistique pour supprimer les variables inutiles. Pour se faire, nous allons réaliser une régression de type Ridge et de type Lasso

```
# install.packages("glmnet")
library(glmnet)
```

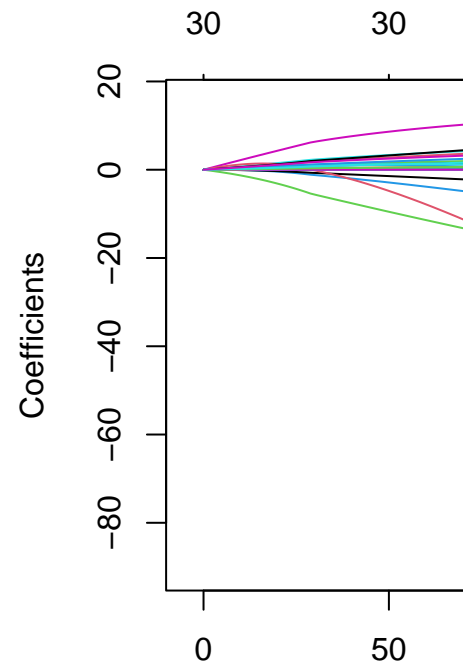
```
## Le chargement a nécessité le package : Matrix
```

```
## Loaded glmnet 4.1-8
```

```
ridge_model <- glmnet(as.matrix(train_data[, -1]), train_data$diagnosis, alpha = 0, family = "binomial")
lasso_model <- glmnet(as.matrix(train_data[, -1]), train_data$diagnosis, alpha = 1, family = "binomial")
```

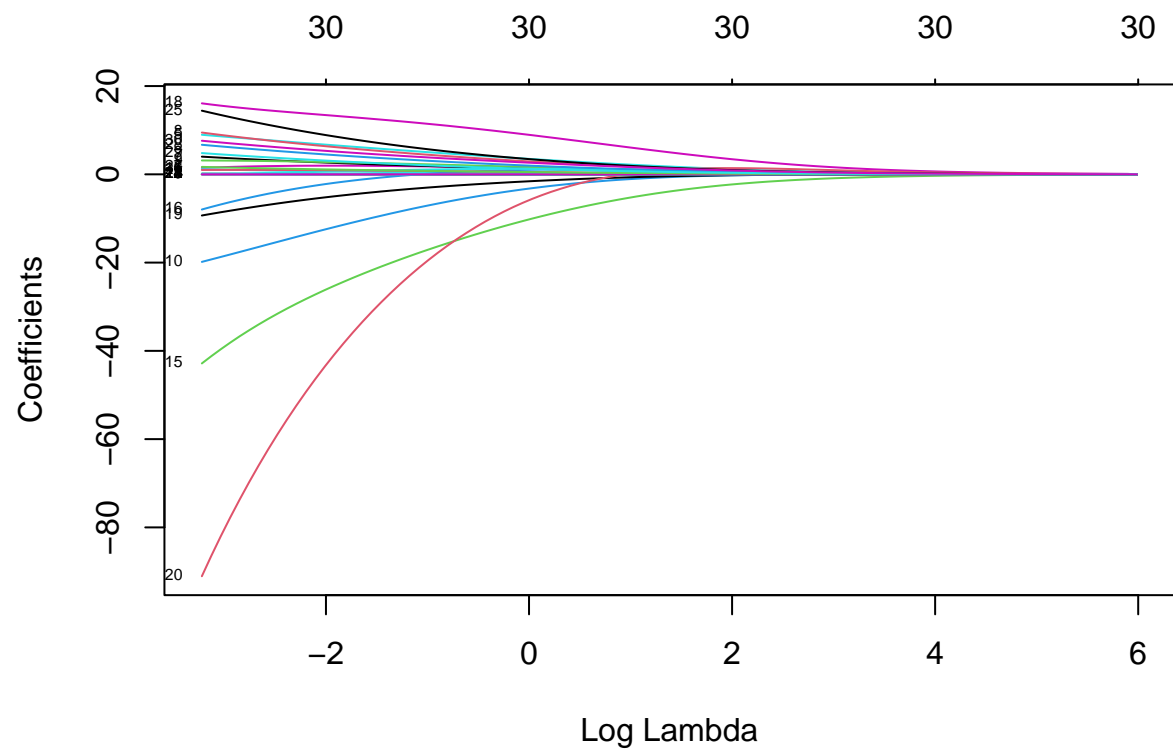
Lancement des algorithmes L'algorithme a bien convergé.

```
plot(ridge_model, label = TRUE)
```

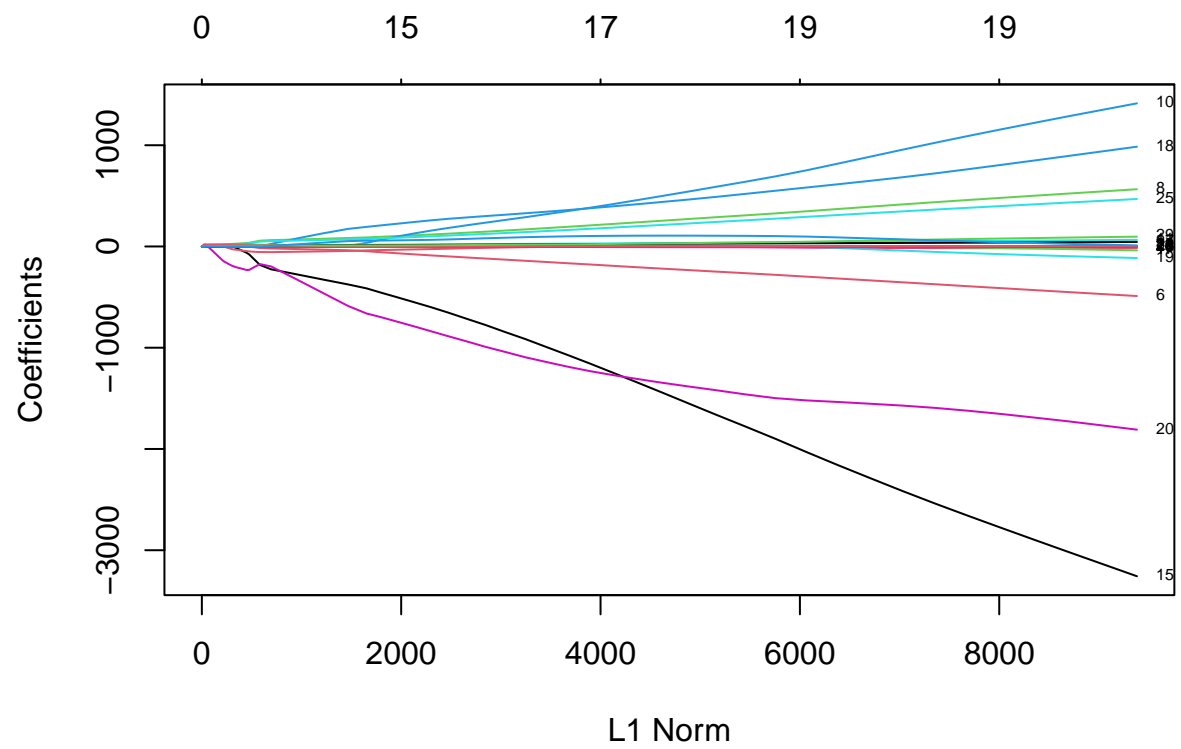


Visualisation des chemins de régularisation des estimateurs ridge et lasso

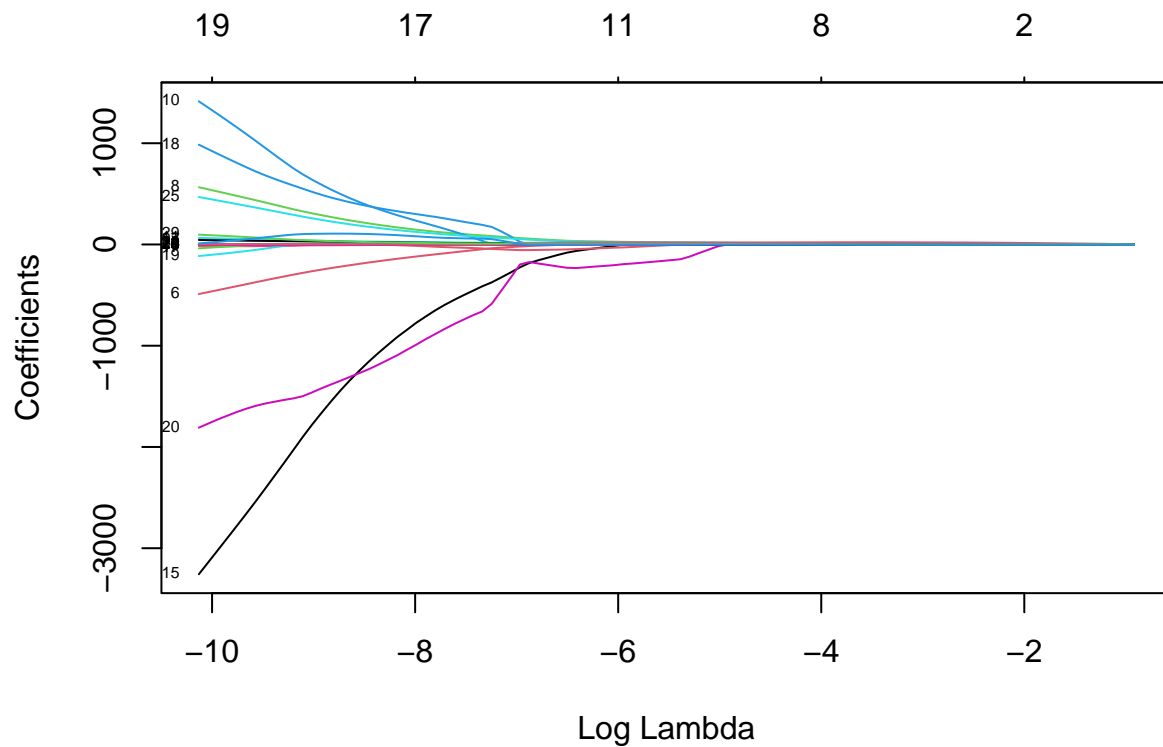
```
plot(ridge_model, xvar = "lambda", label = TRUE)
```



```
plot(lasso_model, label = TRUE)
```



```
plot(lasso_model, xvar = "lambda", label = TRUE)
```



```
pred_ridge <- predict(ridge_model, s = 0.01, newx = as.matrix(test_data[, -1]), type = "response")
pred_ridge_qual <- ifelse(pred_ridge > 0.5, "M", "B")
table(pred_ridge_qual, test_data$diagnosis)
```

Prédictions et performances

```
##
## pred_ridge_qual  B  M
##                B 81  6
##                M  0 27
```

```
ridge_accuracy <- mean(pred_ridge_qual == test_data$diagnosis) # accuracy
ridge_accuracy
```

```
## [1] 0.9473684
```

```
pred_lasso <- predict(lasso_model, s = 0.01, newx = as.matrix(test_data[, -1]), type = "response")
pred_lasso_qual <- ifelse(pred_lasso > 0.5, "M", "B")
table(pred_lasso_qual, test_data$diagnosis)
```

```
##
```



```
## pred_lasso_qual  B  M
##                  B 81 6
##                  M  0 27
```

```
lasso_accuracy <- mean(pred_lasso_qual == test_data$diagnosis) # accuracy
lasso_accuracy
```

```
## [1] 0.9473684
```

```
roc_ridge <- roc(test_data$diagnosis, pred_ridge)
```

```
## Setting levels: control = B, case = M
```

```
## Warning in roc.default(test_data$diagnosis, pred_ridge): Deprecated use a
## matrix as predictor. Unexpected results may be produced, please pass a numeric
## vector.
```

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

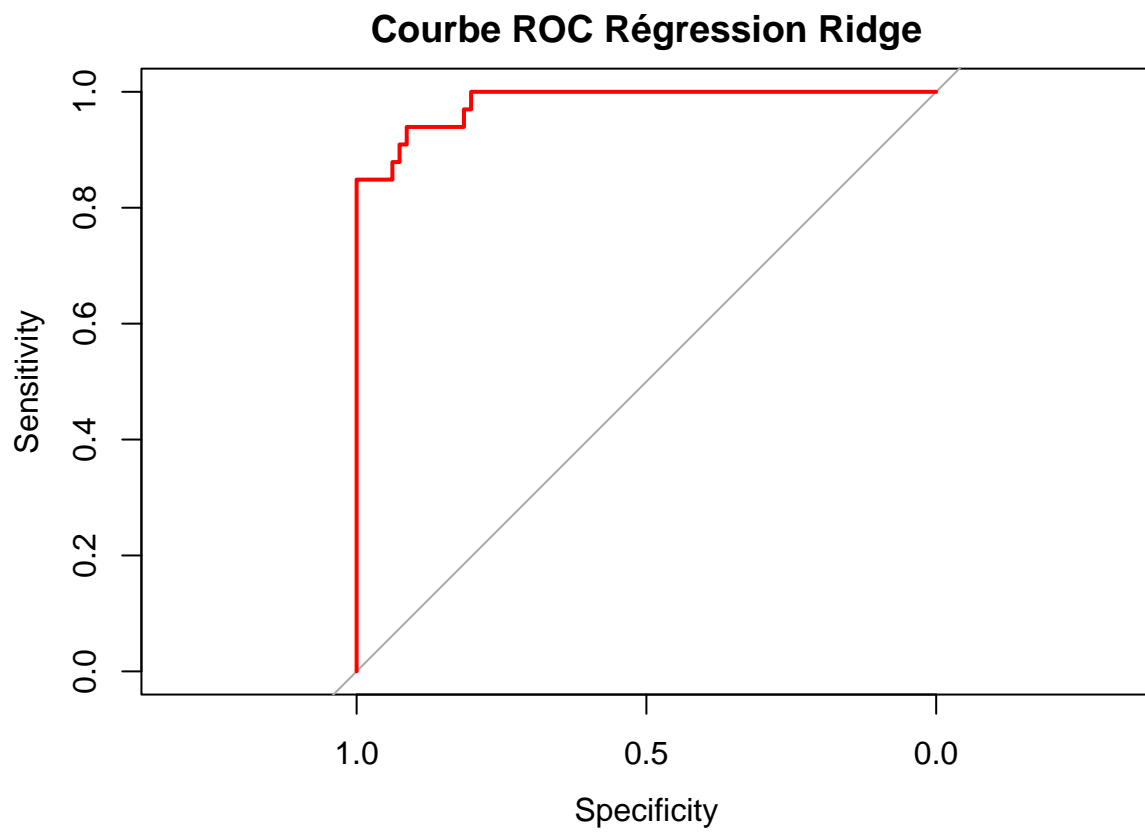
```
roc_lasso <- roc(test_data$diagnosis, pred_lasso)
```

```
## Setting levels: control = B, case = M
```

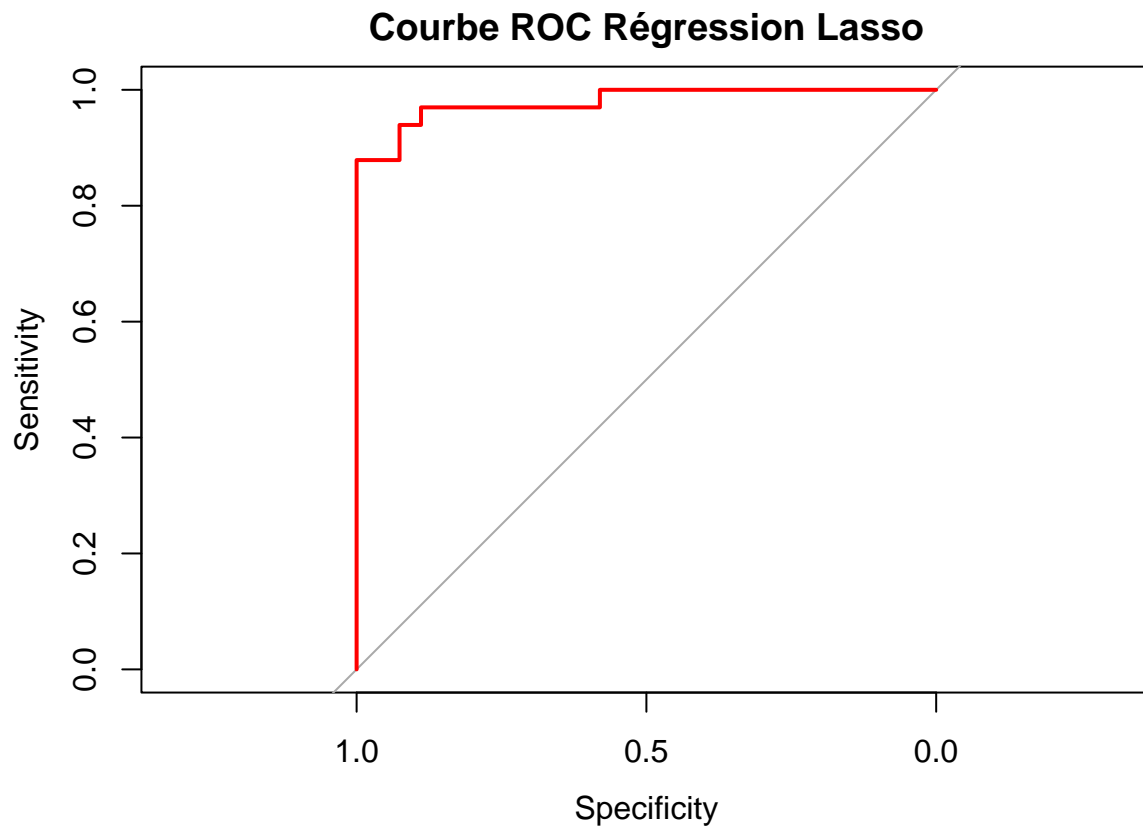
```
## Warning in roc.default(test_data$diagnosis, pred_lasso): Deprecated use a
## matrix as predictor. Unexpected results may be produced, please pass a numeric
## vector.
```

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

```
plot(roc_ridge, col = "red", AUC = TRUE, main = "Courbe ROC Régression Ridge")
```



```
plot(roc_lasso, col = "red", AUC = TRUE, main = "Courbe ROC Régression Lasso")
```



```
auc_ride <- auc(roc_ride)
auc_lasso <- auc(roc_lasso)
auc_ride
```

```
## Area under the curve: 0.9817
```

```
auc_lasso
```

```
## Area under the curve: 0.9794
```

```
sum(coef(lasso_model, s=exp(-6))!=0)
```

Nombre de variables sélectionnées

```
## [1] 12
```

```
sum(coef(ridge_model, s=exp(-6))!=0)
```

```
## [1] 31
```

On a sélectionné beaucoup moins de variables pour le modèle Lasso, qui a des performances similaires.

SVM

Nous avons décidé d'expérimenter un modèle SVM pour voir si les performances sont meilleures, comme nous l'avons fait lors de nos projets industriels.

Lancement du modèle

```
# install.packages("e1071")
library(e1071)
```

```
svm_lin_model <- svm(diagnosis ~ ., data = train_data, kernel = "linear", probability = TRUE )
svm_rbf_model <- svm(diagnosis ~ ., data = train_data, kernel = "radial", probability = TRUE )
```

Prédiction

```
pred_svm_lin <- predict(svm_lin_model, test_data, probability = TRUE)
pred_svm_lin_prob <- attr(pred_svm_lin, "probabilities")
table(pred_svm_lin, test_data$diagnosis)
```

```
##
## pred_svm_lin  B  M
##              B 79  5
##              M  2 28
```

```
pred_svm_rbf <- predict(svm_rbf_model, test_data, probability = TRUE)
pred_svm_rbf_prob <- attr(pred_svm_rbf, "probabilities")
table(pred_svm_rbf, test_data$diagnosis)
```

```
##
## pred_svm_rbf  B  M
##              B 77  3
##              M  4 30
```

Performance du modèle

```
svm_lin_accuracy <- mean(pred_svm_lin == test_data$diagnosis) # accuracy
svm_lin_accuracy
```

```
## [1] 0.9385965
```

```
svm_rbf_accuracy <- mean(pred_svm_rbf == test_data$diagnosis) # accuracy
svm_rbf_accuracy
```

```
## [1] 0.9385965
```

```
roc_svm_lin <- roc(test_data$diagnosis, pred_svm_lin_prob[,2])
```

```
## Setting levels: control = B, case = M
```

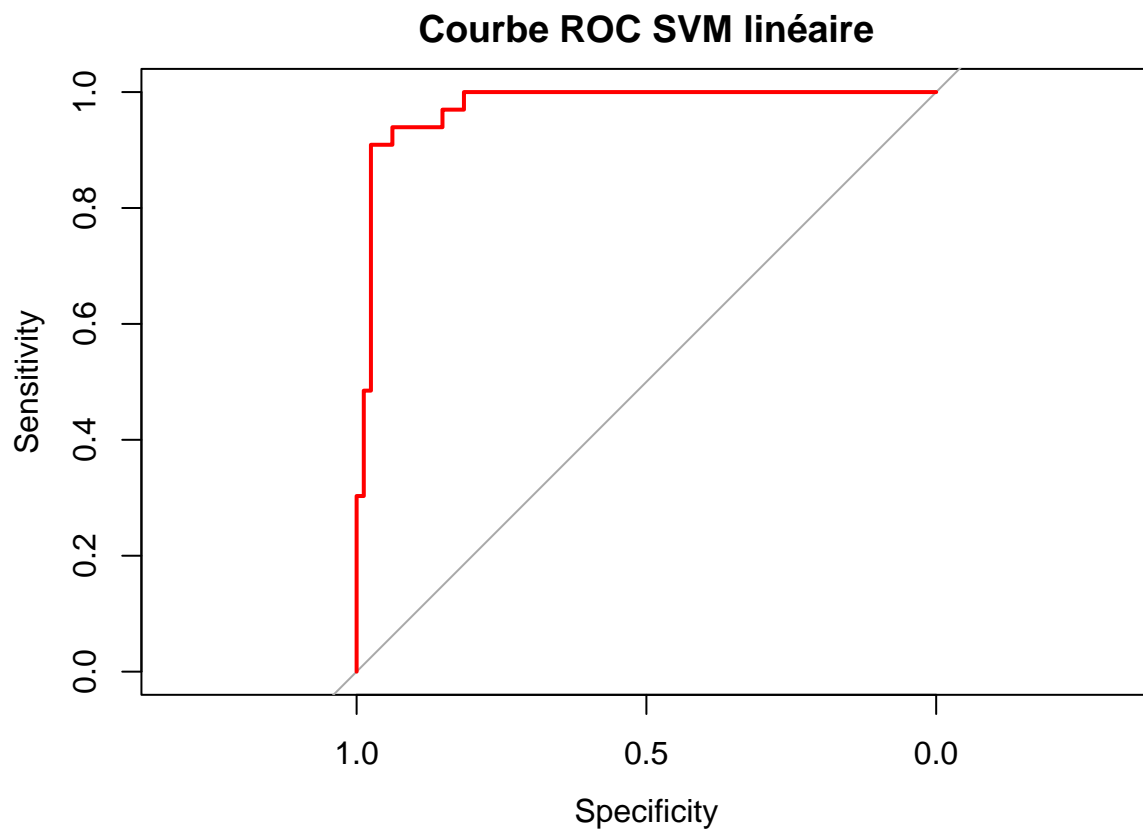
```
## Setting direction: controls > cases
```

```
roc_svm_rbf <- roc(test_data$diagnosis, pred_svm_rbf_prob[,2])
```

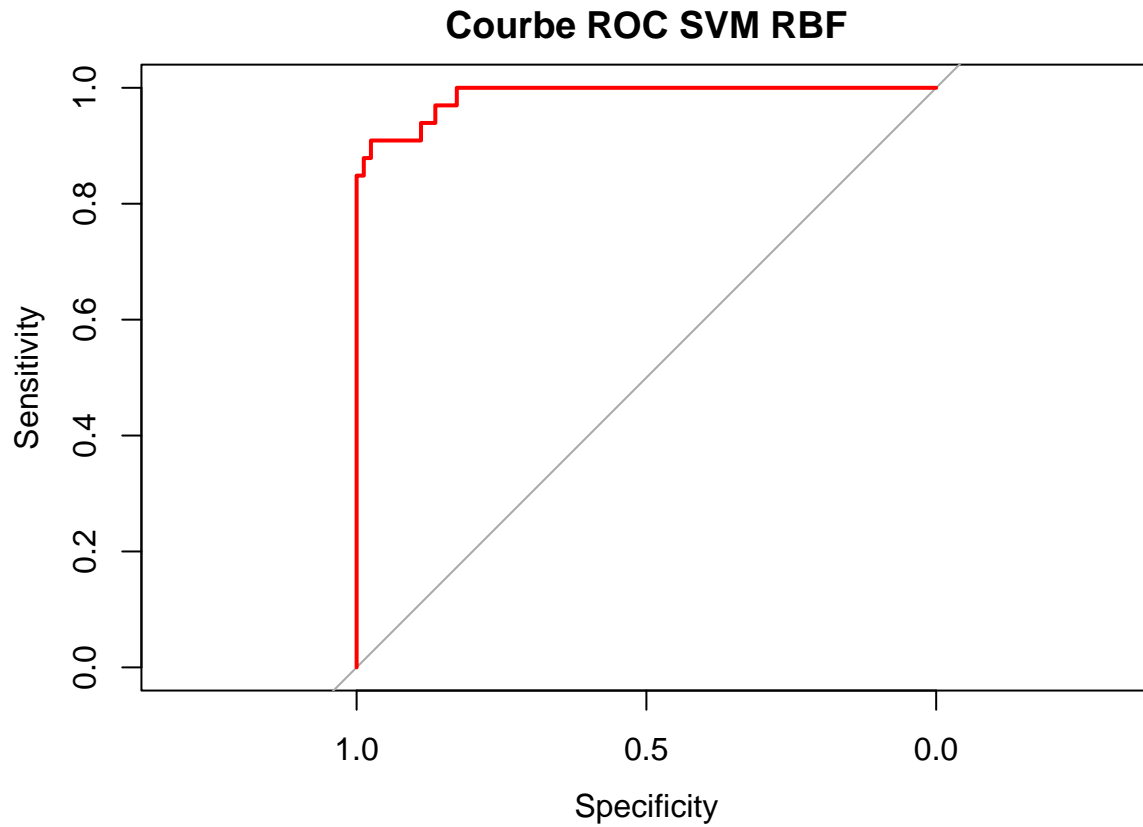
```
## Setting levels: control = B, case = M
```

```
## Setting direction: controls > cases
```

```
plot(roc_svm_lin, col = "red", AUC = TRUE, main = "Courbe ROC SVM linéaire")
```



```
plot(roc_svm_rbf, col = "red", AUC = TRUE, main = "Courbe ROC SVM RBF")
```



```
auc_svm_lin <- auc(roc_svm_lin)
auc_svm_rbf <- auc(roc_svm_rbf)
auc_svm_lin
```

```
## Area under the curve: 0.9753
```

```
auc_svm_rbf
```

```
## Area under the curve: 0.9862
```

On obtient des AUC très satisfaisantes.

ADABOOST

Nous avons décidé d'expérimenter un modèle ADABOOST pour voir si les performances sont meilleures, comme nous l'avons fait lors de nos projets industriels.

Lancement du modèle

```
# install.packages("gbm")
library(gbm)
```

```
## Loaded gbm 2.1.9
```

```
## This version of gbm is no longer under development. Consider transitioning to gbm3, https://github.com
```

On lance un modèle non-calibré pour voir les performances de base.

```
ada_model <- gbm(as.numeric(diagnosis)-1 ~ ., data = train_data, distribution = "adaboost")
ada_model
```

```
## gbm(formula = as.numeric(diagnosis) - 1 ~ ., distribution = "adaboost",
##      data = train_data)
## A gradient boosted model with adaboost loss function.
## 100 iterations were performed.
## There were 30 predictors of which 15 had non-zero influence.
```

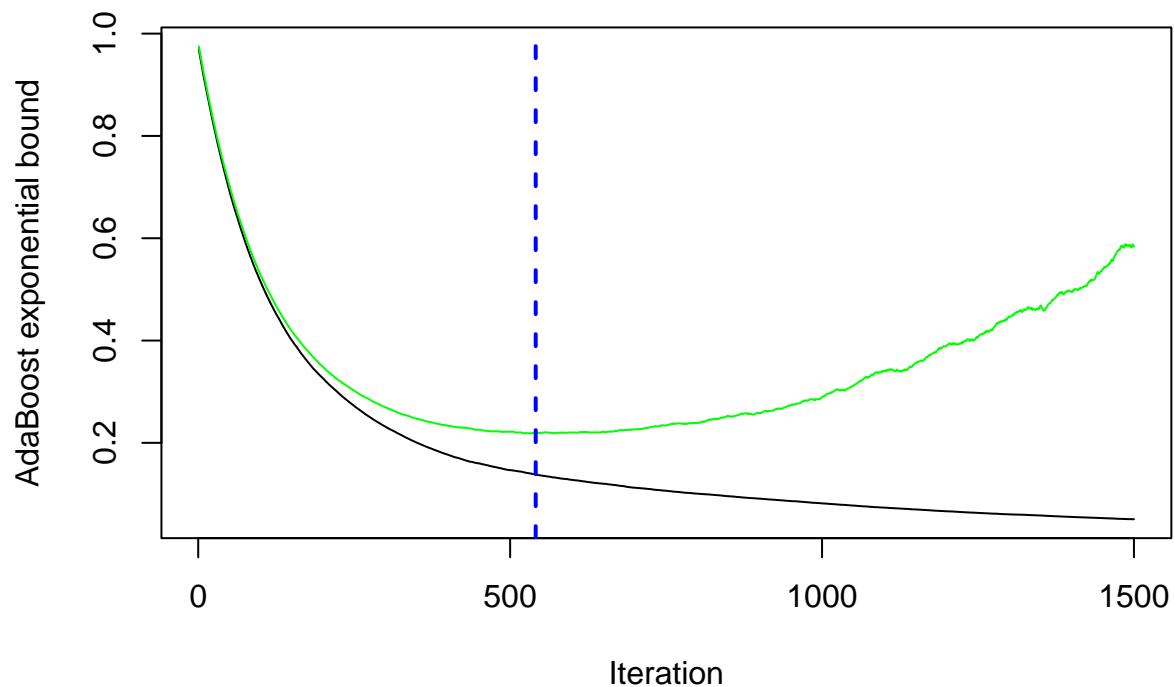
Calibration du modèle

```
ada_cv_adadist_model <- gbm(as.numeric(diagnosis)-1 ~ ., train_data, distribution = "adaboost", cv.folds = 5)
ada_cv_adadist_model
```

Calibration pour une distribution adaboost

```
## gbm(formula = as.numeric(diagnosis) - 1 ~ ., distribution = "adaboost",
##      data = train_data, n.trees = 1500, shrinkage = 0.01, cv.folds = 5)
## A gradient boosted model with adaboost loss function.
## 1500 iterations were performed.
## The best cross-validation iteration was 541.
## There were 30 predictors of which 16 had non-zero influence.
```

```
B.opt <- gbm.perf(ada_cv_adadist_model, method = "cv")
```



En noir, on a la perte pour le jeu "données", et en vert la perte estimée par validation croisée (c'est l'erreur de généralisation). Le trait vertical en pointillé donne le nombre d'itérations qui minimise la perte estimée par validation croisée. Si on prend un nombre d'itérations plus grand, on va faire du sur-apprentissage : on apprendra très bien sur le jeu de données, mais le modèle qu'on aura appris ne va pas bien se généraliser à de nouvelles données.

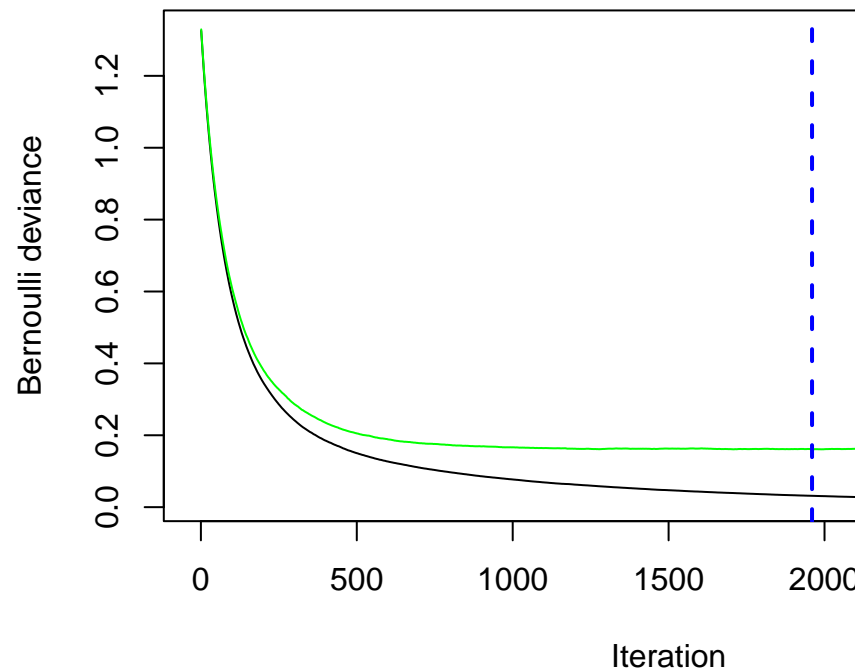
```
B.opt
```

```
## [1] 541
```

On remarque que l'erreur de généralisation tend à augmenter avec le nombre d'itérations. Cela signifie que le modèle fait du sur-apprentissage. La valeur optimale obtenue pour le nombre d'itérations est de 548.

```
ada_cv_berndist_model <- gbm(as.numeric(diagnosis)~1 ~., train_data, distribution = "bernoulli",cv.fold
```

```
B.opt <- gbm.perf(ada_cv_berndist_model, method = "cv")
```

Calibration pour une distribution bernoulli

```
B.opt
```

```
## [1] 1960
```

On obtient une valeur optimale de 1035 pour le nombre d'itérations.

Prédiction

```
pred_ada <- predict(ada_model, test_data, n.trees = 548)
```

ADABOOST sans cross-validation

```
## Warning in predict.gbm(ada_model, test_data, n.trees = 548): Number of trees
## not specified or exceeded number fit so far. Using 100.
```

```
pred_ada_qual <- ifelse(pred_ada > 0.5, "M", "B")
table(pred_ada_qual, test_data$diagnosis)
```

```
##
## pred_ada_qual  B  M
##               B 79  8
##               M  2 25
```

```
ada_accuracy <- mean(pred_ada_qual == test_data$diagnosis) # accuracy
ada_accuracy
```

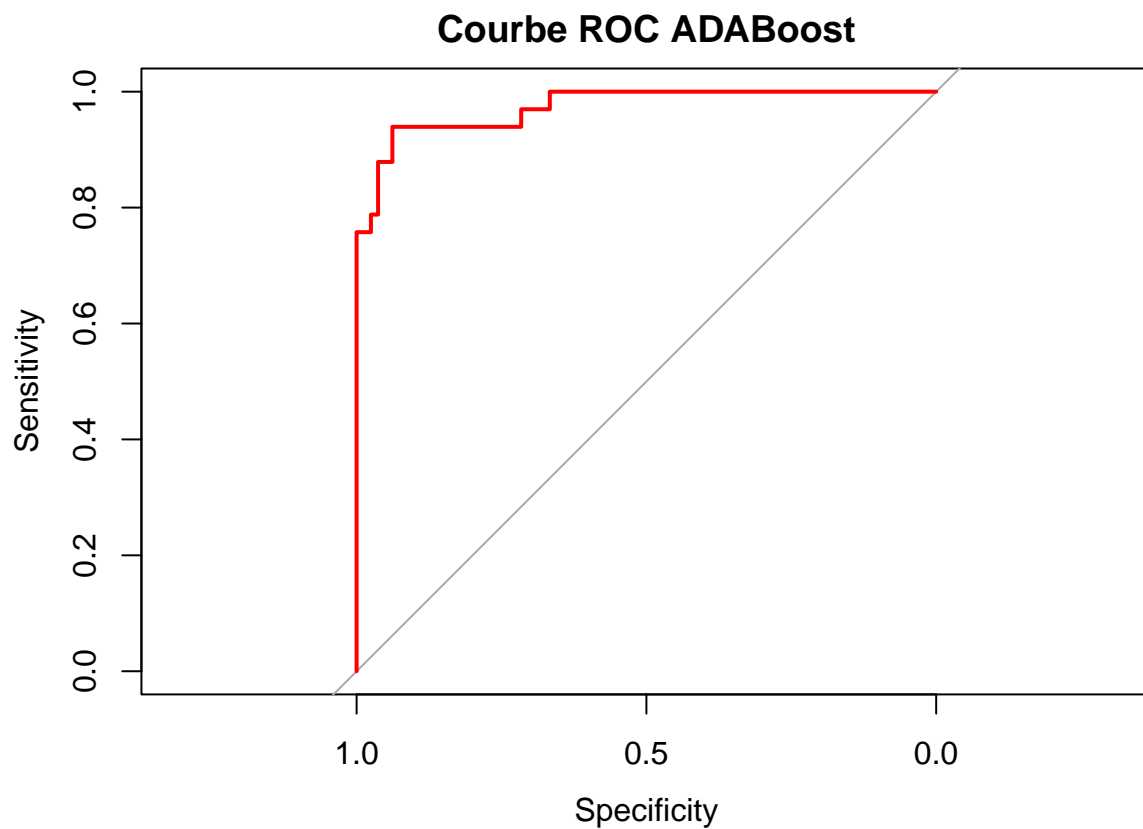
```
## [1] 0.9122807
```

```
roc_ada <- roc(test_data$diagnosis, pred_ada)
```

```
## Setting levels: control = B, case = M
```

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

```
plot(roc_ada, col = "red", AUC = TRUE, main = "Courbe ROC ADABOOST")
```



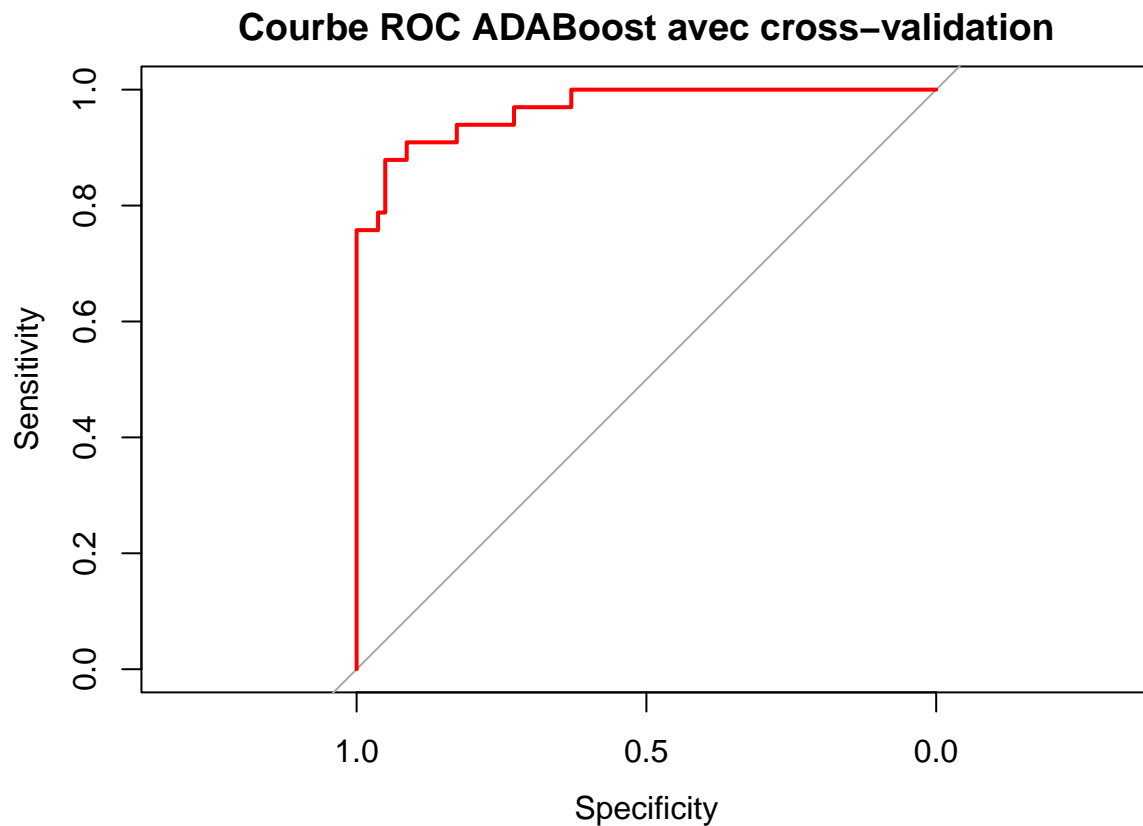
```
auc_ada <- auc(roc_ada)
auc_ada
```

```
## Area under the curve: 0.9734
```

```
pred_ada_cv_adadist <- predict(ada_cv_adadist_model, test_data, n.trees = 548)
pred_ada_cv_adadist_qual <- ifelse(pred_ada_cv_adadist > 0.5, "M", "B")
table(pred_ada_cv_adadist_qual, test_data$diagnosis)
```

ADABOOST avec cross-validation, distribution adaboost

```
##  
## pred_ada_cv_adadist_qual  B  M  
##                          B 79  8  
##                          M  2 25  
  
ada_cv_adadist_accuracy <- mean(pred_ada_cv_adadist_qual == test_data$diagnosis) # accuracy  
ada_cv_adadist_accuracy  
  
## [1] 0.9122807  
  
roc_ada_cv_adadist <- roc(test_data$diagnosis, pred_ada_cv_adadist)  
  
## Setting levels: control = B, case = M  
  
## Setting direction: controls < cases  
  
plot(roc_ada_cv_adadist, col = "red", AUC = TRUE, main = "Courbe ROC ADABOOST avec cross-validation")
```



```
auc_ada_cv_adadist <- auc(roc_ada_cv_adadist)  
auc_ada_cv_adadist
```

```
## Area under the curve: 0.9671
```

```

pred_ada_cv_berndist <- predict(ada_cv_berndist_model, test_data, n.trees = 1035)
pred_ada_cv_berndist_qual <- ifelse(pred_ada_cv_berndist > 0.5, "M", "B")
table(pred_ada_cv_berndist_qual, test_data$diagnosis)

```

ADABOOST avec cross-validation, distribution Bernoulli

```

##
## pred_ada_cv_berndist_qual  B  M
##                          B 79  7
##                          M  2 26

ada_cv_berndist_accuracy <- mean(pred_ada_cv_berndist_qual == test_data$diagnosis) # accuracy
ada_cv_berndist_accuracy

```

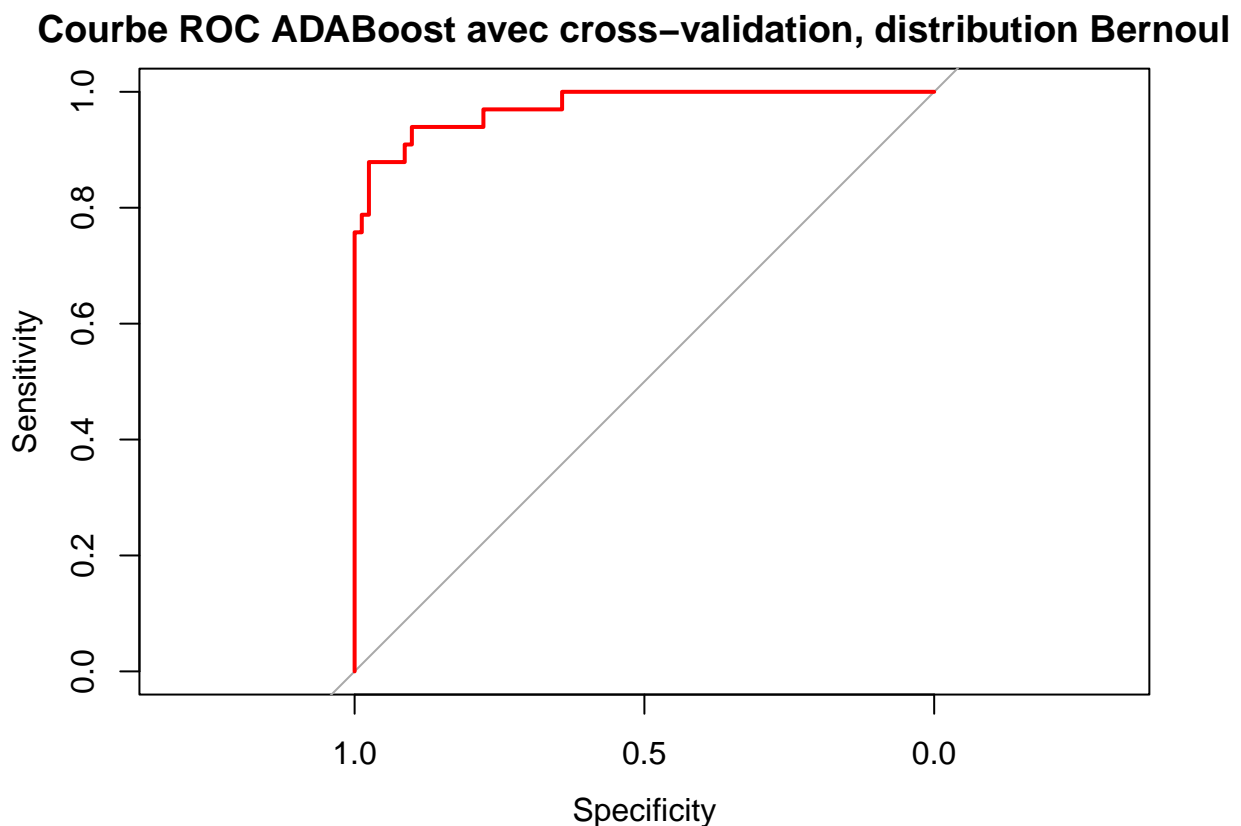
```
## [1] 0.9210526
```

```
roc_ada_cv_berndist <- roc(test_data$diagnosis, pred_ada_cv_berndist)
```

```
## Setting levels: control = B, case = M
```

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

```
plot(roc_ada_cv_berndist, col = "red", AUC = TRUE, main = "Courbe ROC ADABOOST avec cross-validation, d
```



```
auc_ada_cv_berndist <- auc(roc_ada_cv_berndist)
auc_ada_cv_berndist
```

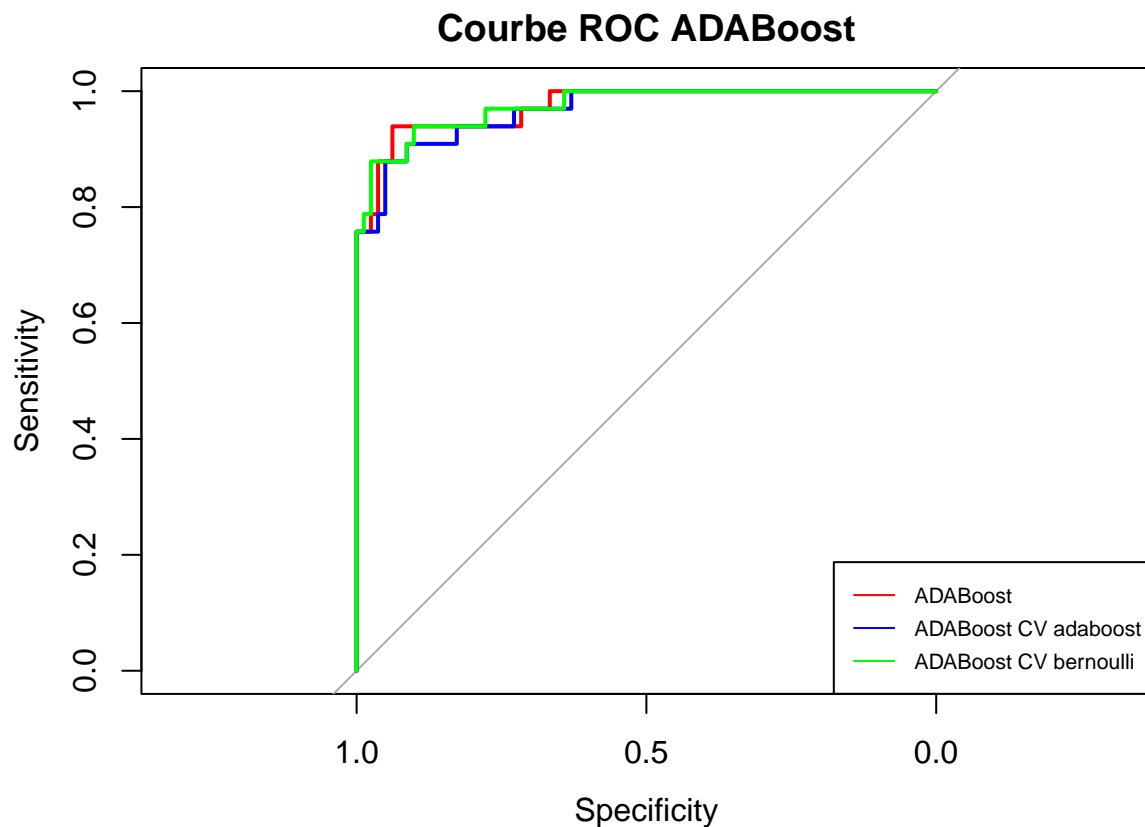
```
## Area under the curve: 0.9742
```

Comparaison des modèles ADABOOST

```
df_ada <- data.frame(model = c("ADABOOST", "ADABOOST CV adaboost", "ADABOOST CV bernoulli"), accuracy =
df_ada
```

```
##           model accuracy      auc
## 1          ADABOOST 0.9122807 0.9734381
## 2 ADABOOST CV adaboost 0.9122807 0.9670782
## 3 ADABOOST CV bernoulli 0.9210526 0.9741863
```

```
plot(roc_ada, col = "red", AUC = TRUE, main = "Courbe ROC ADABOOST")
plot(roc_ada_cv_adadist, col = "blue", AUC = TRUE, add = TRUE)
plot(roc_ada_cv_berndist, col = "green", AUC = TRUE, add = TRUE)
legend("bottomright", legend = c("ADABOOST", "ADABOOST CV adaboost", "ADABOOST CV bernoulli"), col = c(
```



On obtient la même accuracy pour les trois modèles. Cependant, le modèle ADABOOST avec cross-validation et distribution Bernoulli a une AUC légèrement supérieure, et les trois modèles ont des AUC très proches et satisfaisantes.

Evaluation des modèles

Courbes ROC

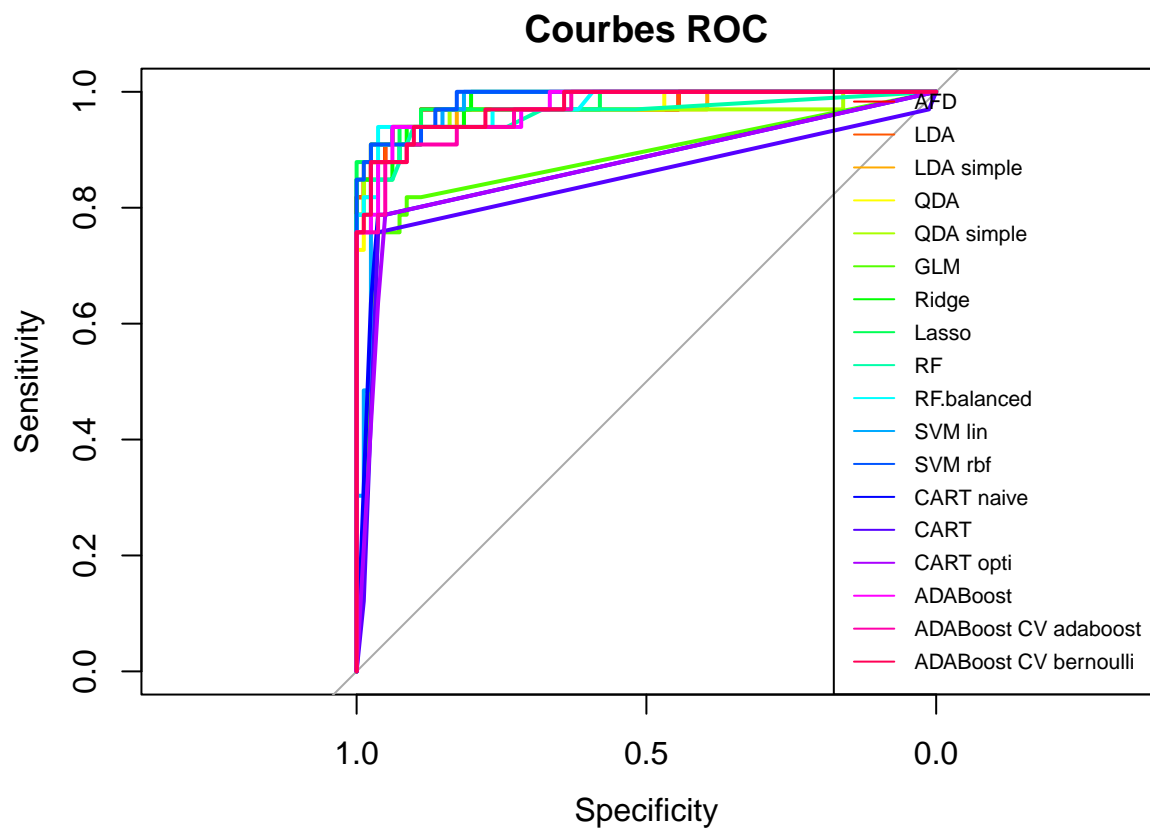
```
# install.packages("pROC")  
library("pROC")
```

Calcul des courbes ROC

```
roc_list <- list(roc_afd, roc_lda, roc_lda_simple, roc_qda, roc_qda_simple, roc_glm, roc_ridge, roc_las  
  
legends_list <- c("AFD", "LDA", "LDA simple", "QDA", "QDA simple", "GLM", "Ridge", "Lasso", "RF", "RF.b
```

Affichage des courbes ROC

```
plot(roc_afd, col = "red", main = "Courbes ROC")  
cols <- rainbow(length(roc_list))  
j <- 1  
for (i in roc_list) {  
  plot(i, add = TRUE, col = cols[j], label = legends_list[j])  
  j <- j + 1  
}  
legend("bottomright", legend = legends_list, col = cols, lty = 1, cex = 0.7)
```



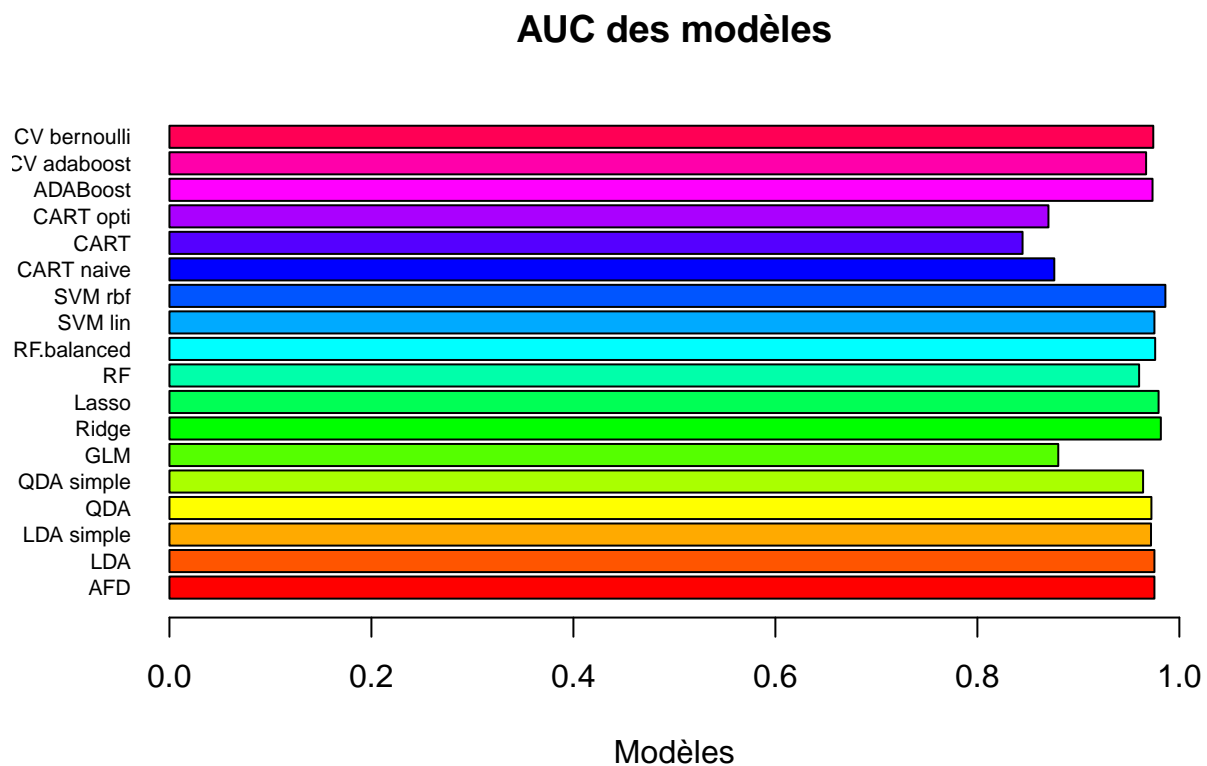
AUC

```
df.auc <- data.frame(model = legends_list, auc = sapply(roc_list, function(x) auc(x)))
df.auc
```

```
##           model      auc
## 1           AFD 0.9753086
## 2           LDA 0.9753086
## 3  LDA simple 0.9719416
## 4           QDA 0.9723158
## 5  QDA simple 0.9640853
## 6           GLM 0.8800973
## 7         Ridge 0.9816685
## 8         Lasso 0.9794239
## 9           RF 0.9601571
## 10  RF.balanced 0.9760569
## 11        SVM lin 0.9753086
## 12        SVM rbf 0.9861579
## 13    CART naive 0.8761691
## 14          CART 0.8447437
## 15    CART opti 0.8703704
## 16      ADABoost 0.9734381
## 17 ADABoost CV adaboost 0.9670782
## 18 ADABoost CV bernoulli 0.9741863
```

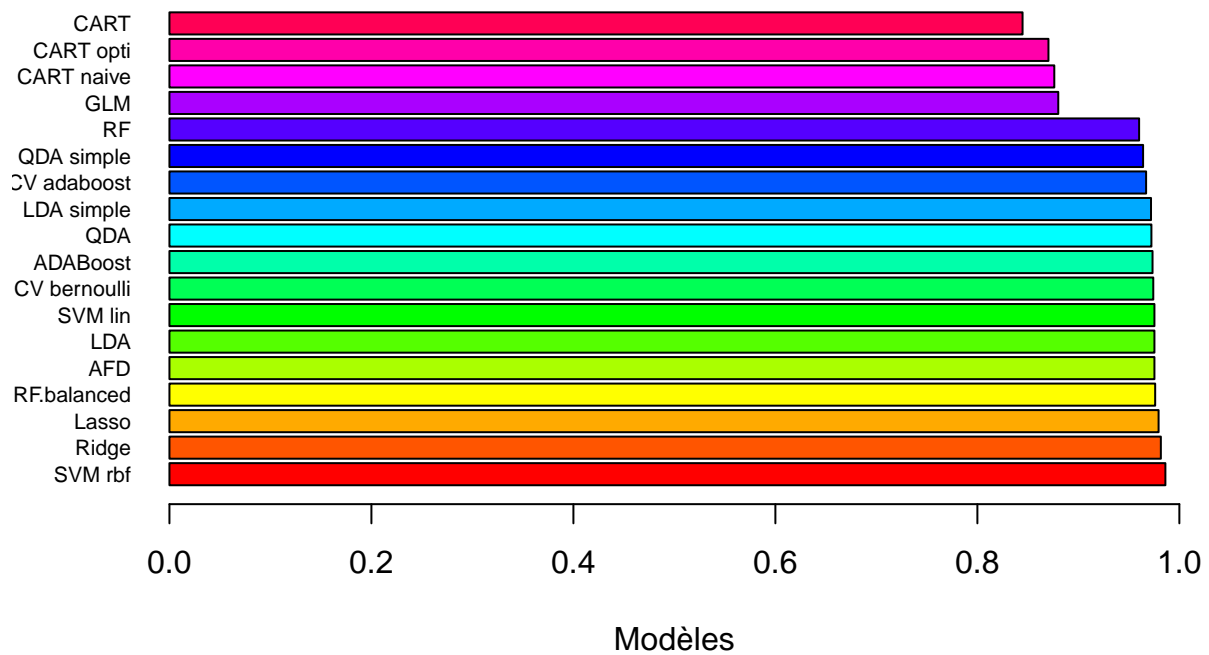
```
# Sauvegarde des AUC dans un fichier csv
write.csv(df.auc, "data/auc.csv")
```

```
barplot(df.auc$auc, names.arg = df.auc$model, col = rainbow(length(df.auc$auc)), main = "AUC des modèles")
```



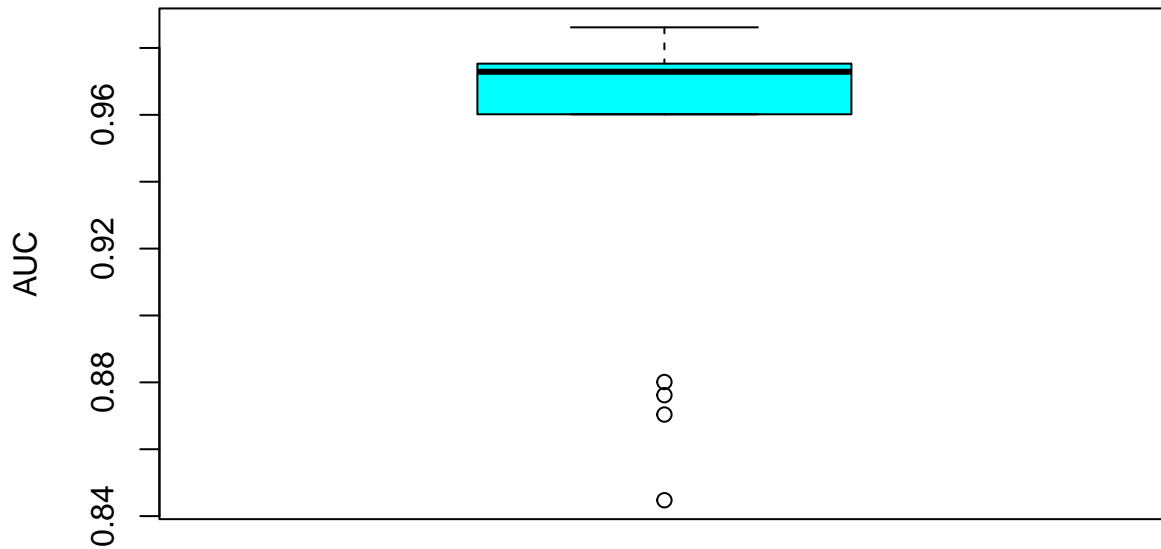
```
df.auc_sorted <- df.auc[order(df.auc$auc, decreasing = TRUE),]
barplot(df.auc_sorted$auc, names.arg = df.auc_sorted$model, col = rainbow(length(df.auc_sorted$auc)), main = "AUC des modèles triés")
```


AUC des modèles



```
boxplot(df.auc$auc, main = "Boxplot des AUC des modèles", ylab = "AUC", col = "cyan")
```

Boxplot des AUC des modèles



```
summary(df.auc$auc)
```

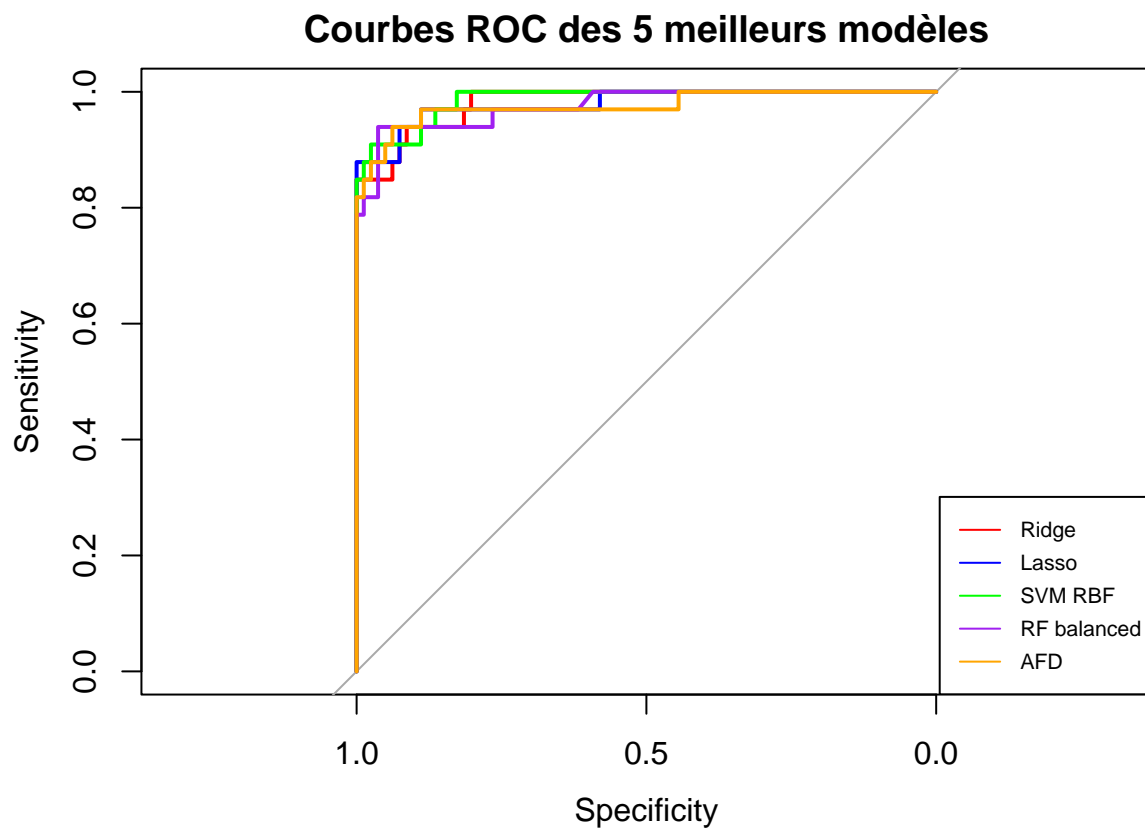
```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.8447  0.9611  0.9729  0.9502  0.9753  0.9862
```

On obtient que le *meilleur modèle* relativement à l'AUC trouvé est le SVM avec noyau RBF (non-linéaire). Le meilleur modèle trouvé que l'on a vu dans ce cours est le modèle de régression logistique avec pénalisation de Ridge, avec une AUC de 0.981. De plus, le troisième meilleur modèle, à savoir la régression Lasso, a une AUC assez proche de celle de la régression Ridge, tout en dépendant de moins de variables.

Les *modèles les moins bons* sont les modèles CART, qui ont des AUC inférieures à 0.9, et de régression logistique non pénalisée, qui ont des performances très inférieures aux autres.

```
# Courbes ROC des 5 meilleurs modèles
```

```
plot(roc_ridge, col = "red", AUC = TRUE, main = "Courbes ROC des 5 meilleurs modèles")
plot(roc_lasso, col = "blue", AUC = TRUE, add = TRUE)
plot(roc_svm_rbf, col = "green", AUC = TRUE, add = TRUE)
plot(roc_rf_balanced, col = "purple", AUC = TRUE, add = TRUE)
plot(roc_afd, col = "orange", AUC = TRUE, add = TRUE)
legend("bottomright", legend = c("Ridge", "Lasso", "SVM RBF", "RF balanced", "AFD"), col = c("red", "blue", "green", "purple", "orange"))
```



Comparaison de l'accuracy des modèles

```
df_accuracy <- data.frame(model = c("AFD", "LDA", "LDA.simple", "QDA", "QDA.simple", "GLM", "Ridge", "L
```

```
df_accuracy
```

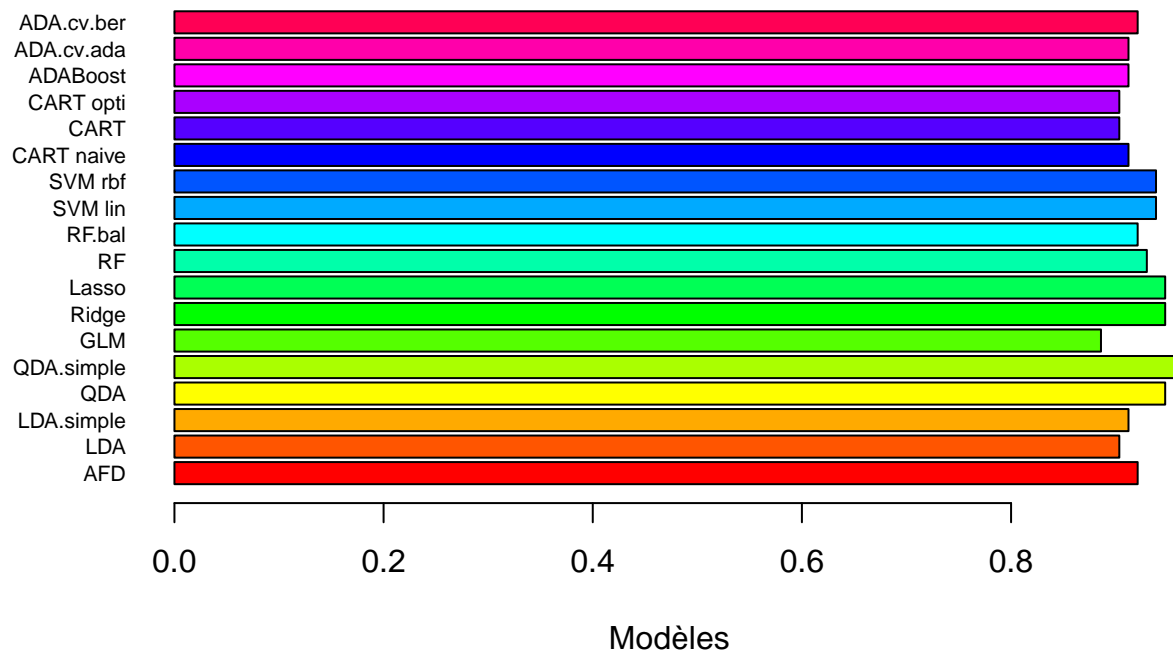
```
##      model accuracy
## 1      AFD 0.9210526
## 2      LDA 0.9035088
## 3 LDA.simple 0.9122807
## 4      QDA 0.9473684
## 5 QDA.simple 0.9561404
## 6      GLM 0.8859649
## 7      Ridge 0.9473684
## 8      Lasso 0.9473684
## 9       RF 0.9298246
## 10 RF.bal 0.9210526
## 11 SVM lin 0.9385965
## 12 SVM rbf 0.9385965
## 13 CART naive 0.9122807
## 14      CART 0.9035088
## 15 CART opti 0.9035088
## 16 ADABoost 0.9122807
```

```
## 17 ADA.cv.ada 0.9122807
## 18 ADA.cv.ber 0.9210526
```

```
# Sauvegarde des accuracy dans un fichier csv
write.csv(df_accuracy, "data/accuracy.csv")
```

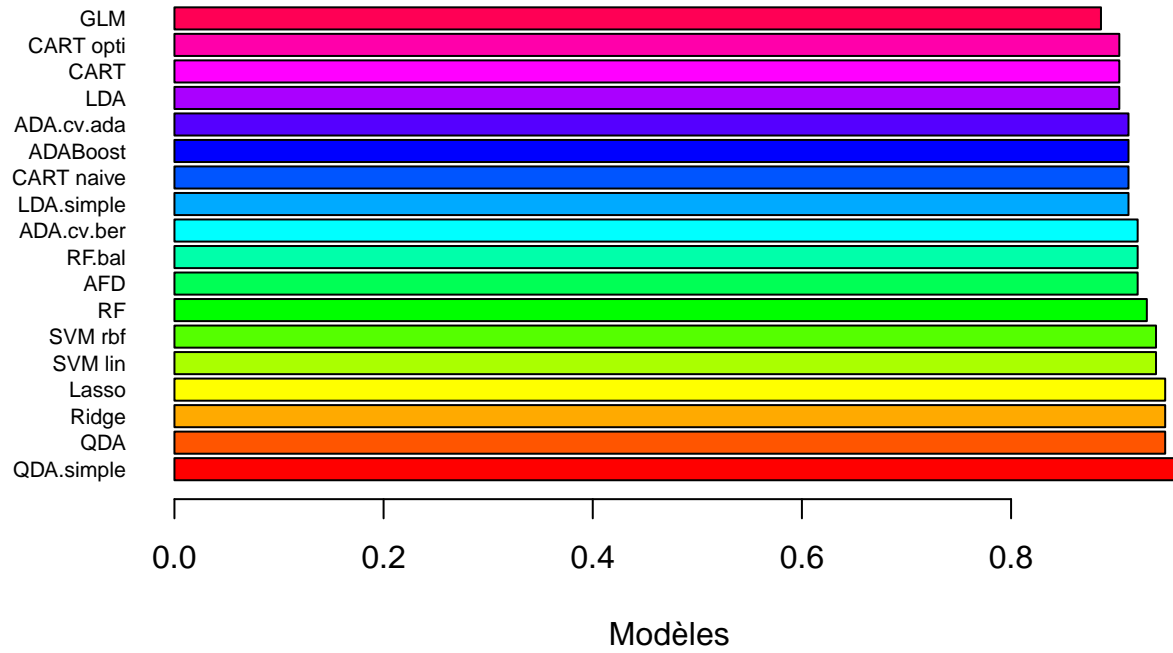
```
barplot(df_accuracy$accuracy, names.arg = df_accuracy$model, col = rainbow(length(df_accuracy$accuracy)))
```

Accuracy des modèles



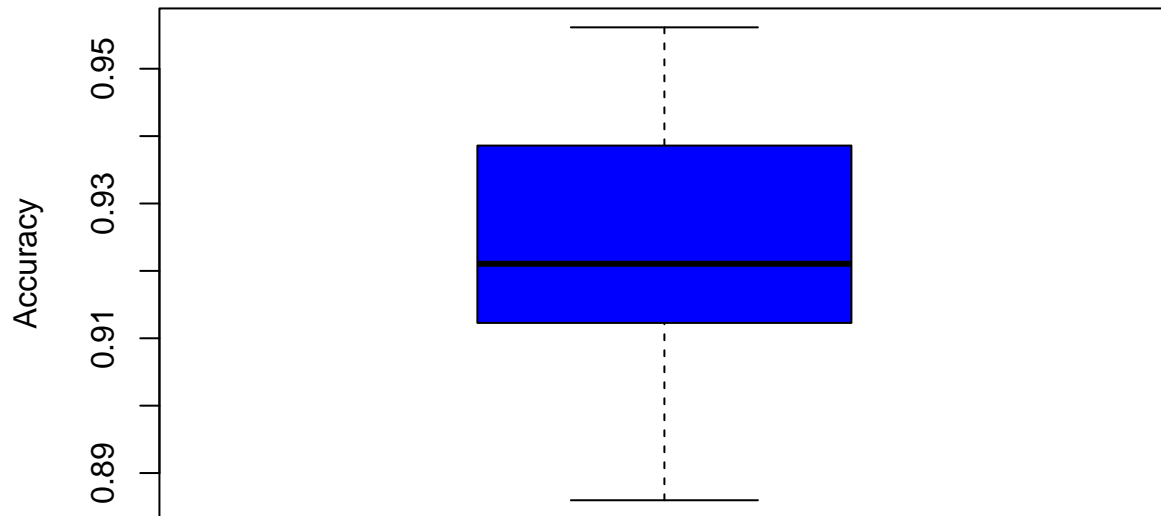
```
df_accuracy_sorted <- df_accuracy[order(df_accuracy$accuracy, decreasing = TRUE),]
barplot(df_accuracy_sorted$accuracy, names.arg = df_accuracy_sorted$model, col = rainbow(length(df_accuracy_sorted$accuracy)))
```

Accuracy des modèles



```
boxplot(df_accuracy$accuracy, main = "Boxplot des accuracy des modèles", ylab = "Accuracy", col = "blue")
```

Boxplot des accuracy des modèles



```
summary(df_accuracy$accuracy)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.8860  0.9123  0.9211  0.9230  0.9386  0.9561
```

- Meilleurs Modèles* : Les modèles avec les meilleures accuracy sont les modèles de QDA, de régression logistique avec pénalisation de Ridge et de régression Lasso. Comme la QDA n'est pas dans les meilleurs modèles en terme d'AUC, on peut dire que les modèles de régression logistique avec pénalisation de Ridge et de régression Lasso sont les meilleurs modèles de statistiques prédictives que l'on a trouvé. Une très bonne accuracy pour la QDA peut être un signe de sur-apprentissage.
- Modèles moins efficaces* : Le modèle qui a le moins fonctionné est le modèle de régression logistique sans pénalisation. En effet, lorsque nous lançons l'analyse, nous obtenions des warnings ce qui signifiait que le modèle n'était pas bien calibré. Le modèle CART a également eu des performances moins intéressantes que l'on a pu observer avec son accuracy plus basse que les autres, tout comme le LDA.

DataFrame du récapitulatif

```
df <- data.frame(
  model = c("AFD", "LDA", "LDA.simple", "QDA", "QDA.simple", "GLM", "Ridge", "Lasso", "RF", "RF.bal", "RF.
  accuracy = c(afd_accuracy, lda_accuracy, lda_simple_accuracy, qda_accuracy, qda_simple_accuracy, glm_
  auc = c(auc_afd, auc_lda, auc_lda_simple, auc_qda, auc_qda_simple, auc_glm, auc_ridge, auc_lasso, auc_
```

```
write.csv(df, "data/summary.csv")
```

```
summary(df)
```

```
##      model          accuracy          auc
## Length:18      Min.    :0.8860      Min.    :0.8447
## Class :character 1st Qu.:0.9123      1st Qu.:0.9611
## Mode  :character Median  :0.9211      Median  :0.9729
##              Mean   :0.9230      Mean   :0.9502
##              3rd Qu.:0.9386      3rd Qu.:0.9753
##              Max.    :0.9561      Max.    :0.9862
```

Interprétation des résultats

Les modèles de régression logistique avec pénalisation de Ridge et de régression Lasso sont les meilleurs modèles de statistiques prédictives que l'on a trouvé. En effet, ils ont les meilleures AUC et les meilleures accuracy. De plus, le modèle de régression Lasso dépend de moins de variables que le modèle de régression Ridge, ce qui peut être un avantage.

Le modèle de SVM avec noyau RBF est le meilleur modèle en terme d'AUC, mais il n'est pas le meilleur en terme d'accuracy.

Le modèle de régression logistique sans pénalisation et le modèle CART sont les moins bons modèles que l'on a testés. Cela peut s'expliquer par une mauvaise calibration pour le modèle de régression logistique.

Conclusion

Le data-set est exploitable en machine learning, et nous avons pu obtenir des modèles de statistiques prédictives qui ont de bonnes performances. Les modèles de régression logistique avec pénalisation de Ridge et de régression Lasso sont les meilleurs modèles que l'on a trouvé. Ces modèles pourraient permettre de prédire le diagnostic de patientes atteintes de cancer du sein, dans une certaine mesure.