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)
```

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 faut positifs. Peut-être que les données d'entrainement 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 réequilibrer 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.

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

```r
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

```r
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.

Simplication du modèle

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

```r
library(klaR)
```

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

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

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

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

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

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

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

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

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

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

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 0.95, ce qui est plutôt bon.

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 = FA
```

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

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

```
##    Length     Class      Mode
##        29      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  4
##   M  2 29
```

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

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

CART

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

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

```r
library(rpart)
```

Lancement du modèle

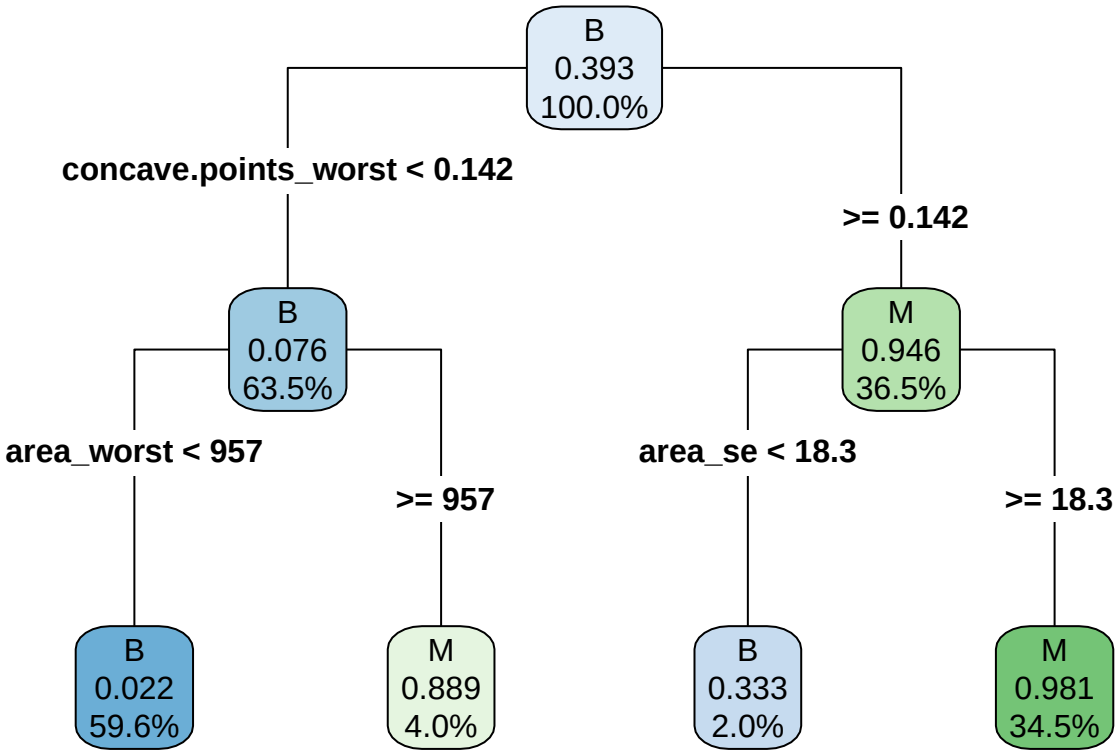
```r
tree <- rpart(diagnosis~., train_data)
```

```r
print(tree)
```

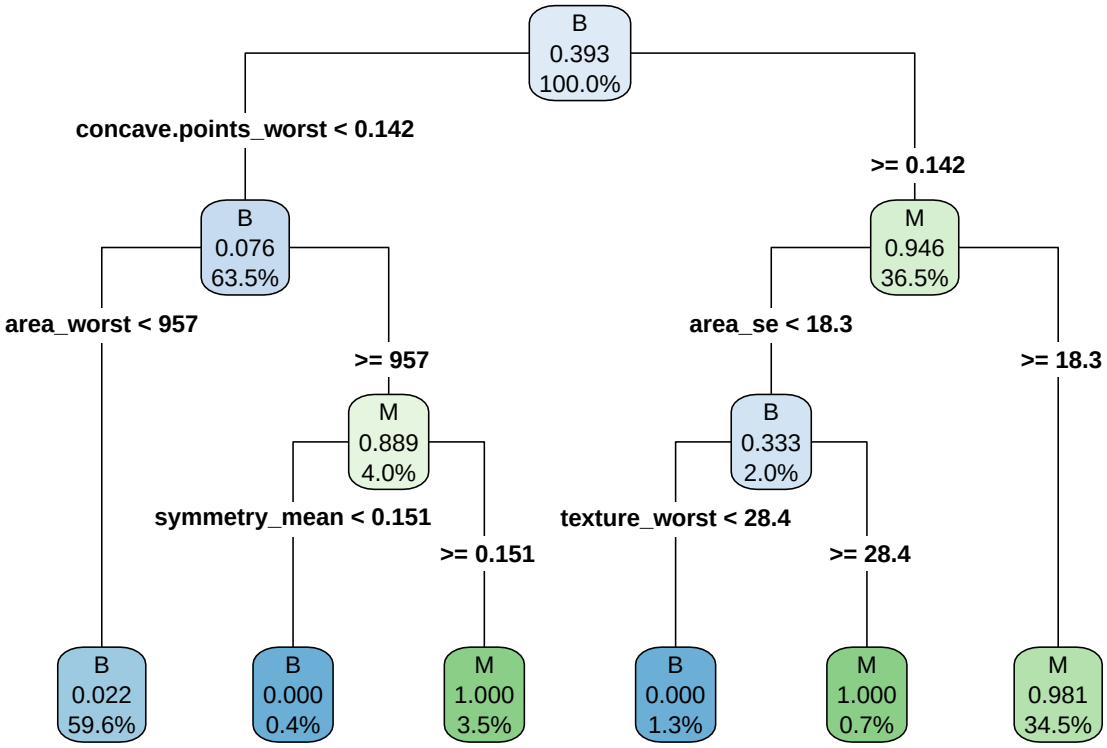
```
## n= 455
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 455 179 B (0.60659341 0.39340659)
##   2) concave.points_worst< 0.14235 289  22 B (0.92387543 0.07612457)
##     4) area_worst< 957.45 271   6 B (0.97785978 0.02214022) *
##     5) area_worst>=957.45 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

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



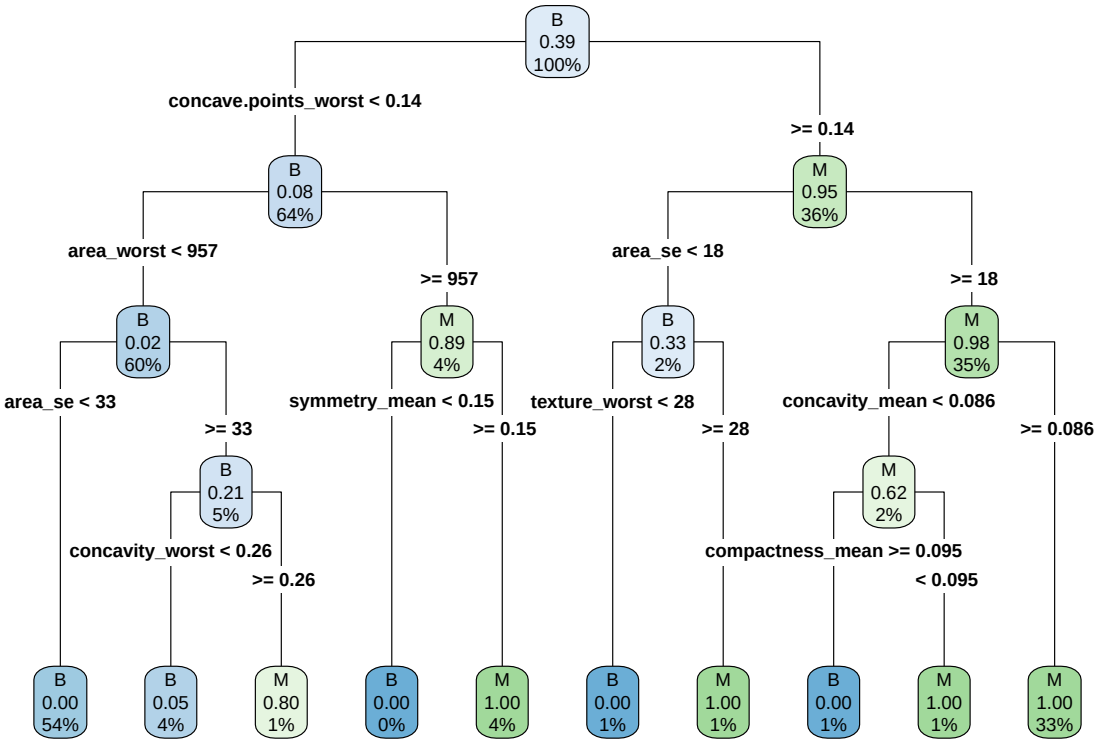

```
tree2 <- rpart(diagnosis~.,train_data,control=rpart.control(minsplit=5))
rpart.plot(tree2, type=4, digits=3)
```



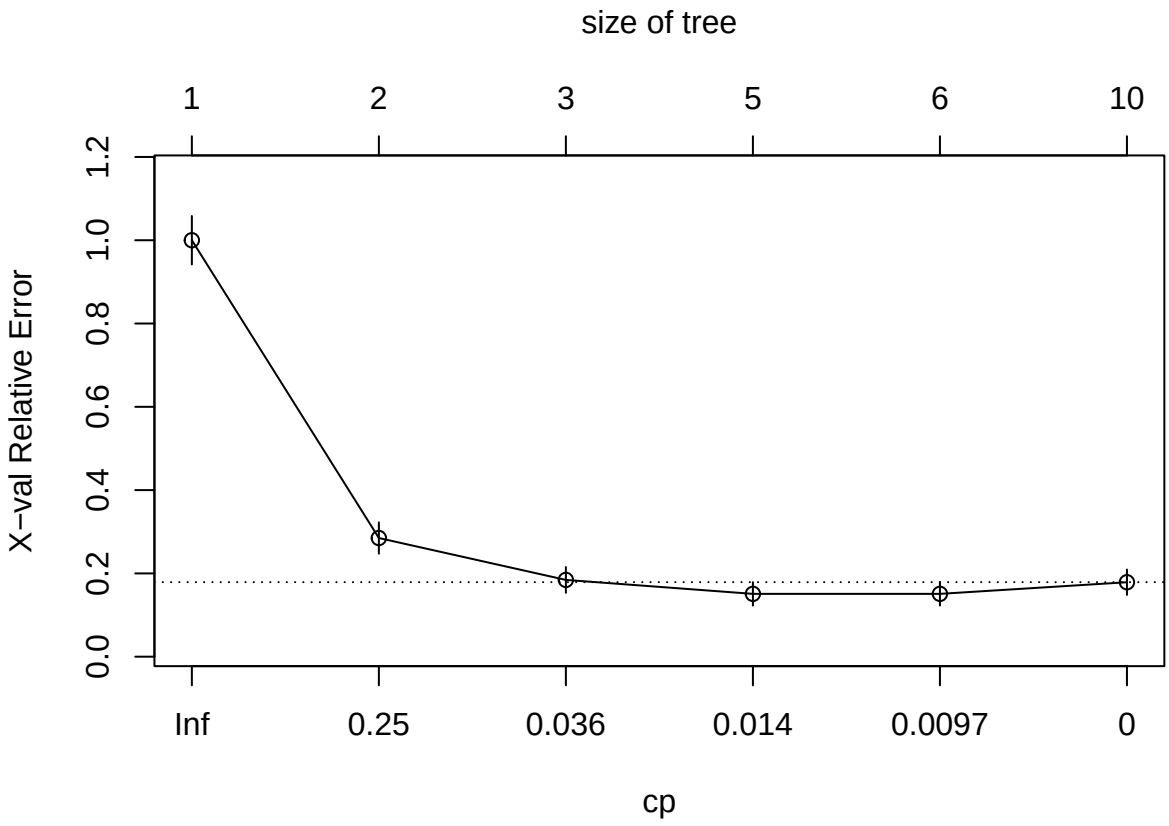
Elagage des arbres

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)
```

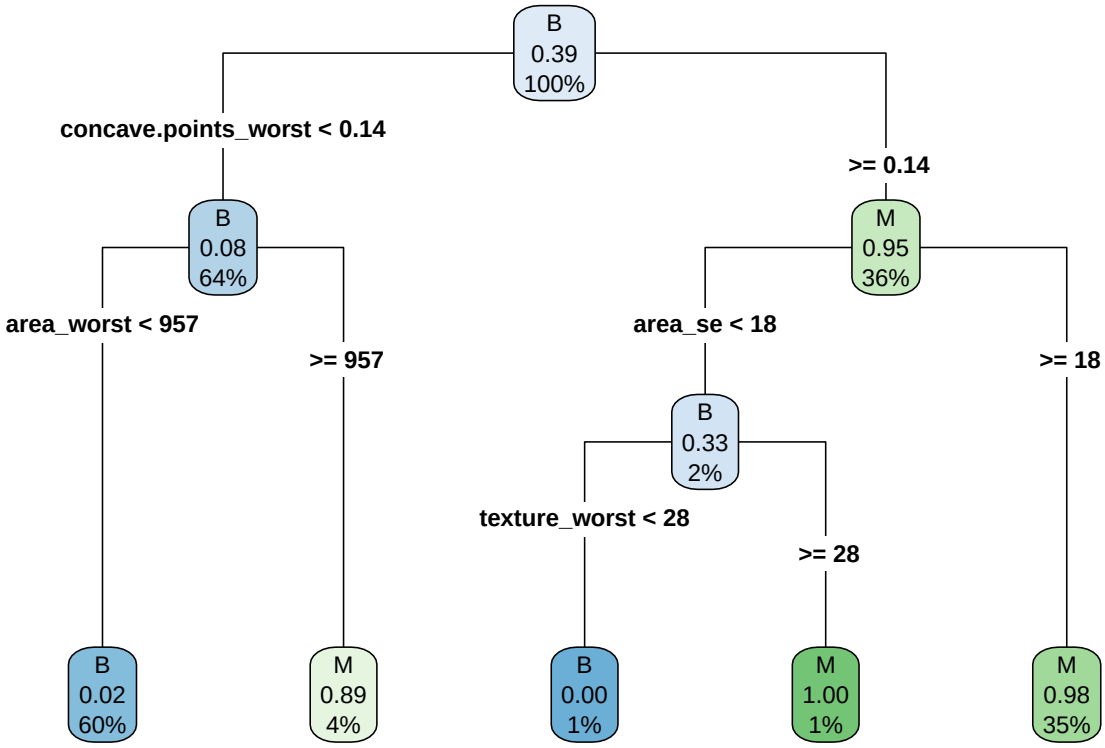



```
plotcp(tree_elag)
```



On observe une erreur minimale pour arbre de taille 4 ou 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 78  7
##                   M  3 26
```

Performance du modèle

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

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

Pour le modèle CART optimal, on obtient une accuracy de 0.90, ce qui est plutôt bon.

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
```

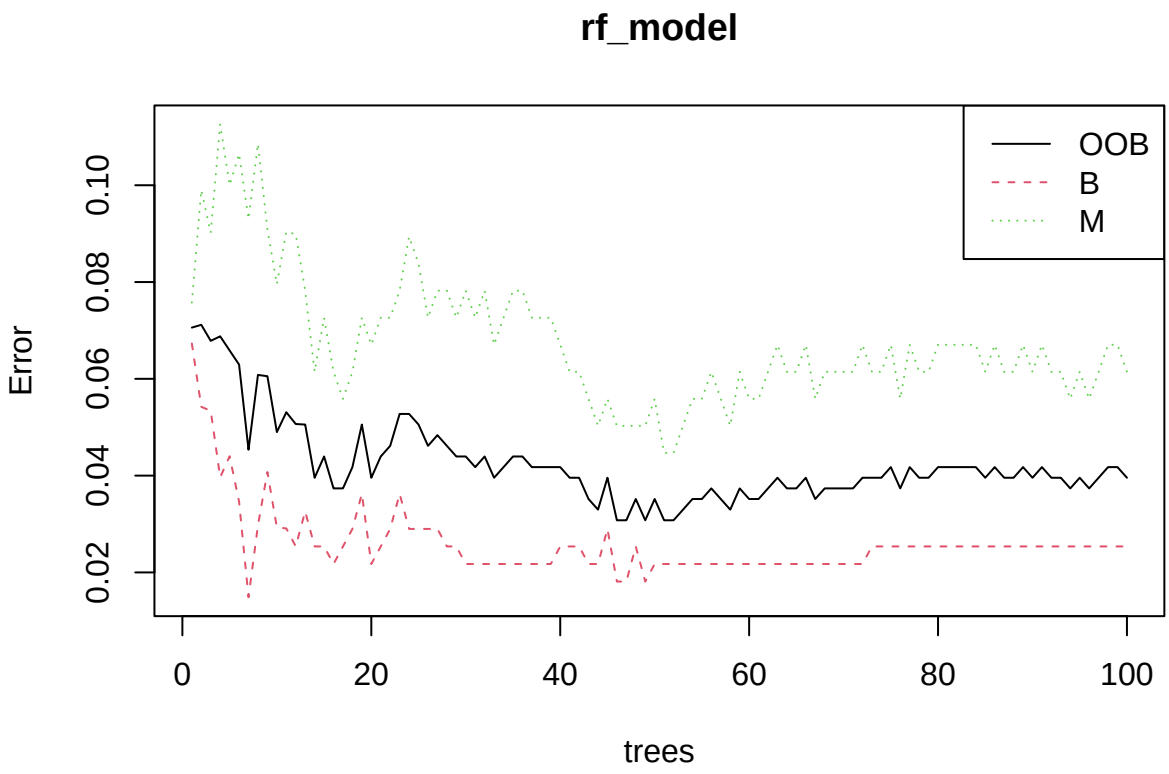
Performance du modèle

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

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

Erreur OOB (Out-of-Bag)

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



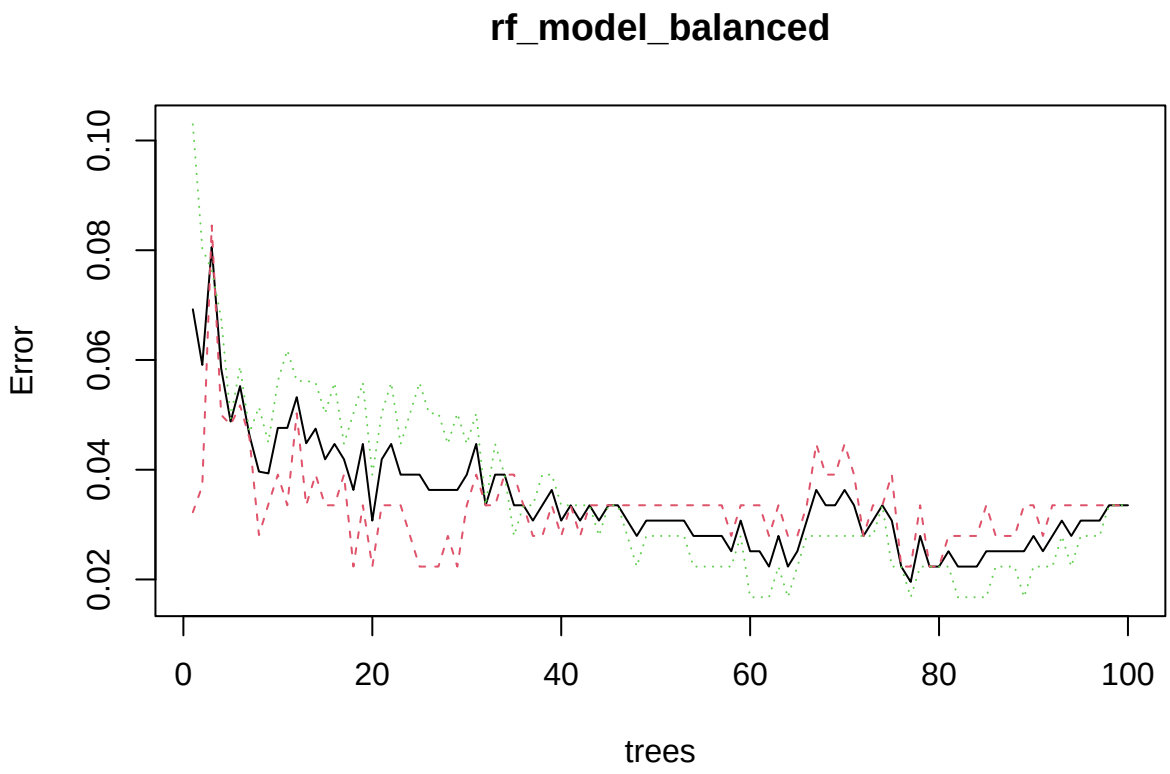
Le taux d'erreur semble s'être stabilisé. On observe que l'erreur OOB pour la classification "M" semble plus élevée que pour la classification "B". Il y a donc un déséquilibre dans les classes. Essayons d'équilibrer le jeu d'entrainement.

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 =
```

```
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  5
##                     M  3 28
```

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

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

On obtient des performances similaires.

Régression Logistique

Lancement du modèle

```r
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.

```r
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.

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)
```

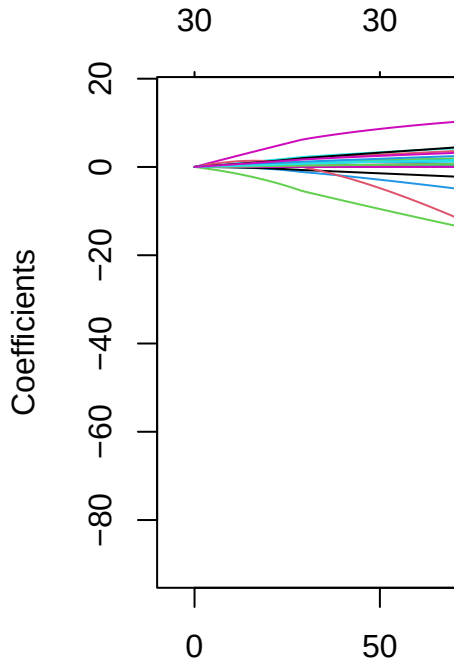
```
## Le chargement a nécessité le package : Matrix
```

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

```r
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"
```

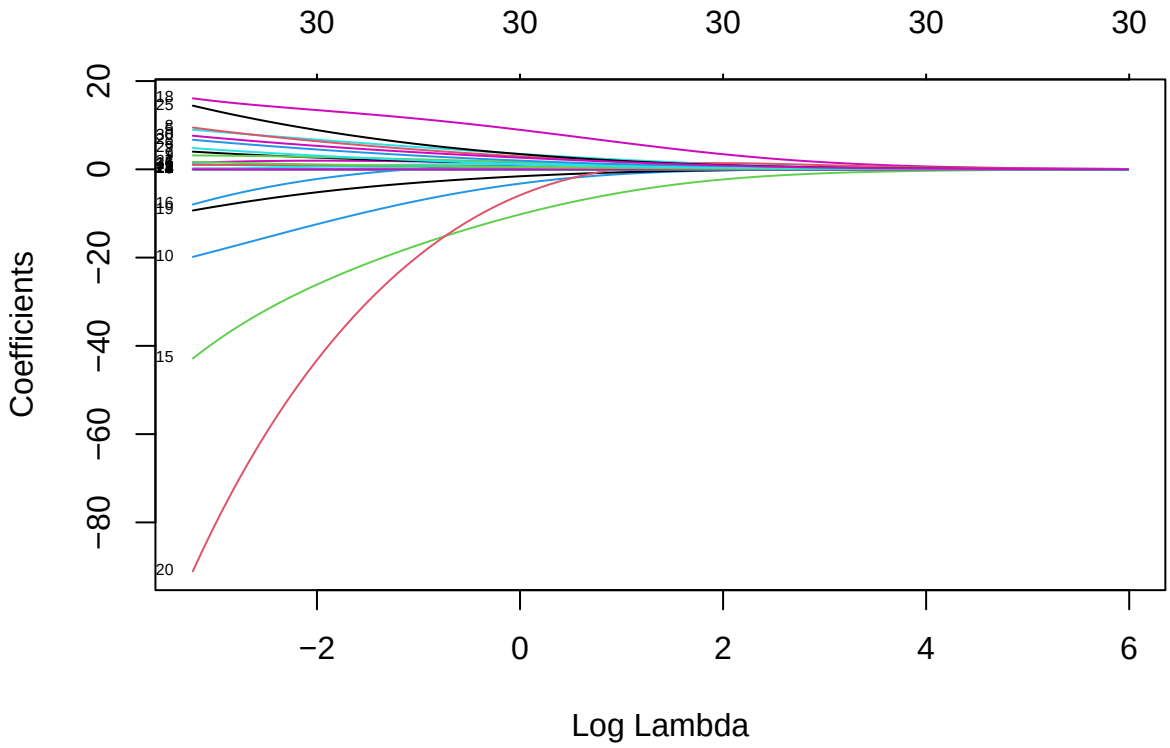
Lancement des algorithmes L'algorithme a bien convergé.

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

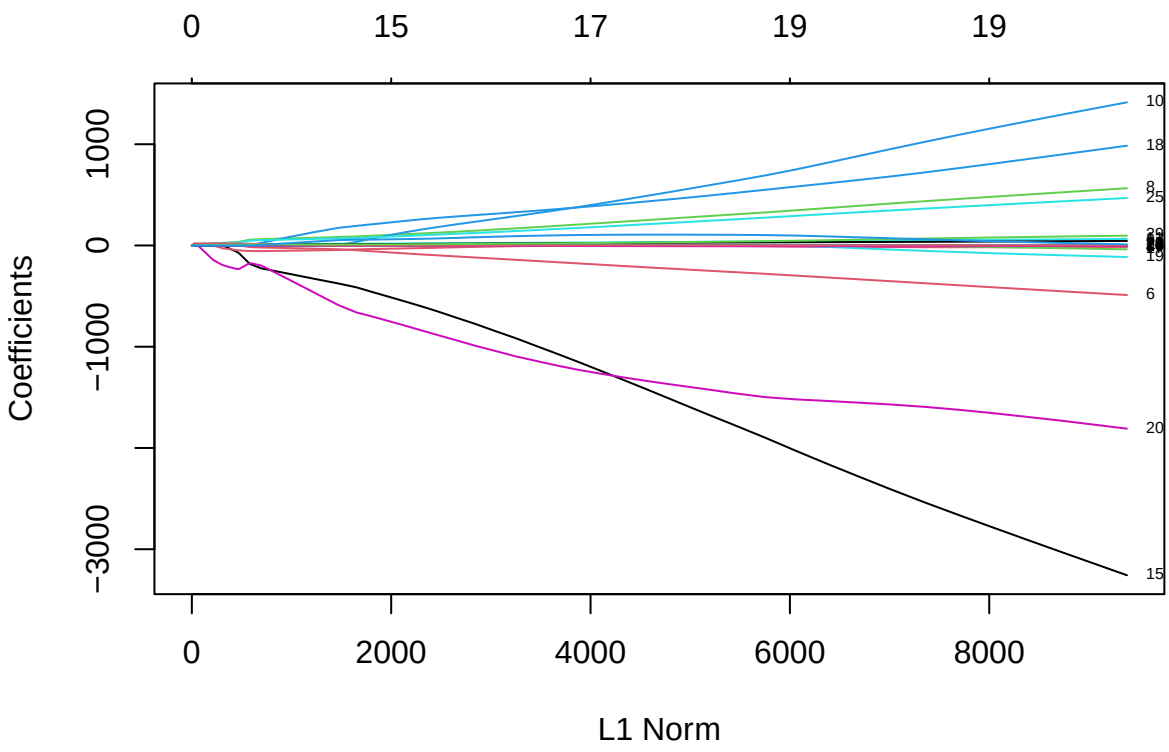


Visualisation des chemins de régularisation des estimateurs `ridge` et `lasso`

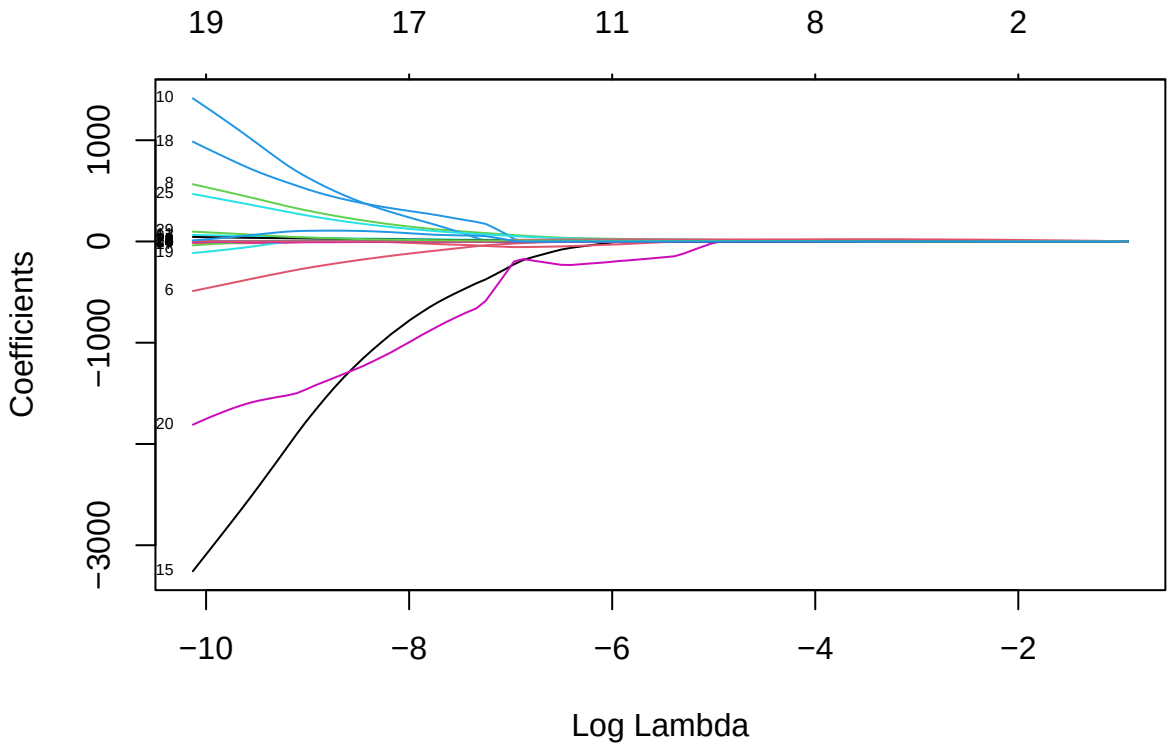
```r
plot(ridge_model, xvar = "lambda", label = TRUE)
```




```
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
```

```
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
```

```r
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

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

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

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

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

Evaluation des modèles

Courbes ROC

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

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

##
## Attachement du package : 'pROC'

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

Calcul des courbes ROC

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

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

## Setting direction: controls < cases
```

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

```
## Setting levels: control = B, case = M
## Setting direction: controls < cases
```

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

```
## Setting levels: control = B, case = M
## Setting direction: controls < cases
```

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

```
## Setting levels: control = B, case = M
## Setting direction: controls < cases
```

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

```
## Setting levels: control = B, case = M
## Setting direction: controls < cases
```

```r
# rpc_cart <- roc(test_data$diagnosis, pred_cart)
roc_glm <- roc(test_data$diagnosis, pred_glm)
```

```
## Setting levels: control = B, case = M
## Setting direction: controls < cases
```

```r
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
```

```r
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
```

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

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

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

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

```
## Setting levels: control = B, case = M
## Setting direction: controls > cases
```

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

```
## Setting levels: control = B, case = M
## Setting direction: controls > cases
```

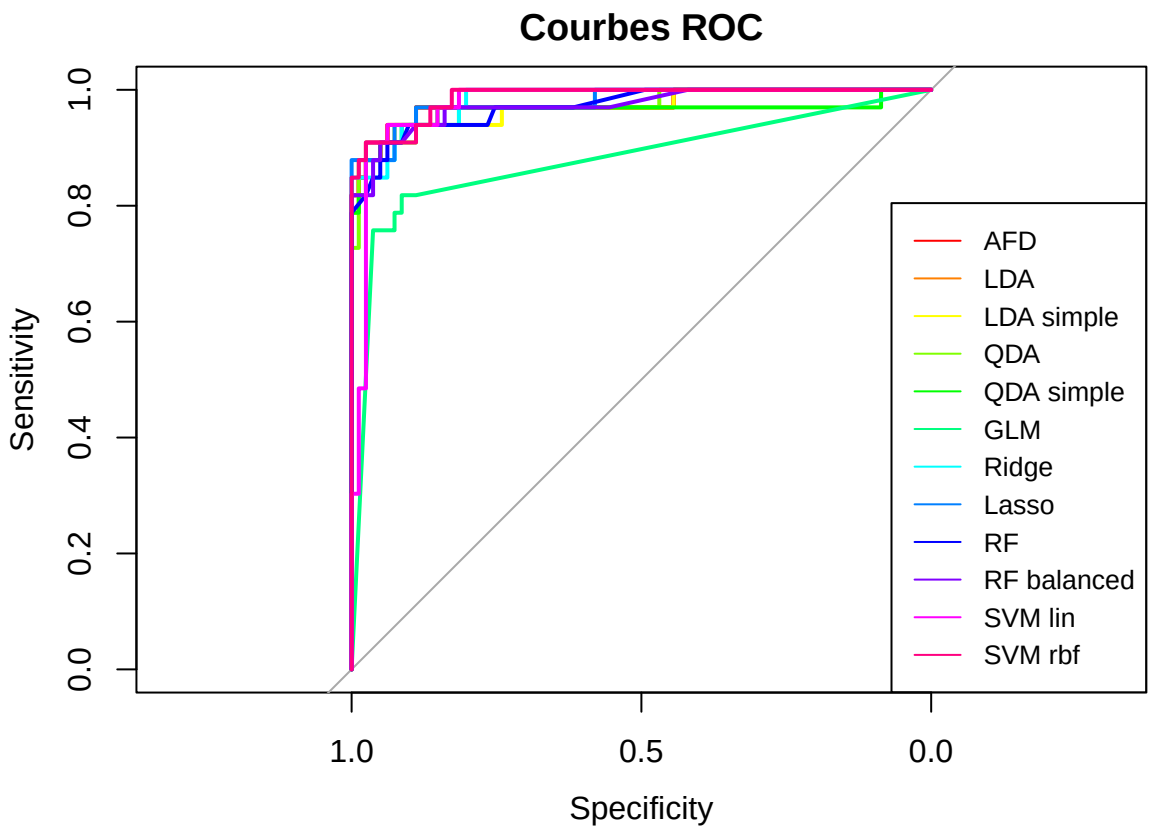
```r
roc_svm_rbf <- roc(test_data$diagnosis, pred_svm_rbf_prob[,2])
```

```
## Setting levels: control = B, case = M
## Setting direction: controls > cases
```

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

Affichage des courbes ROC

```r
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 = 0.8)
```

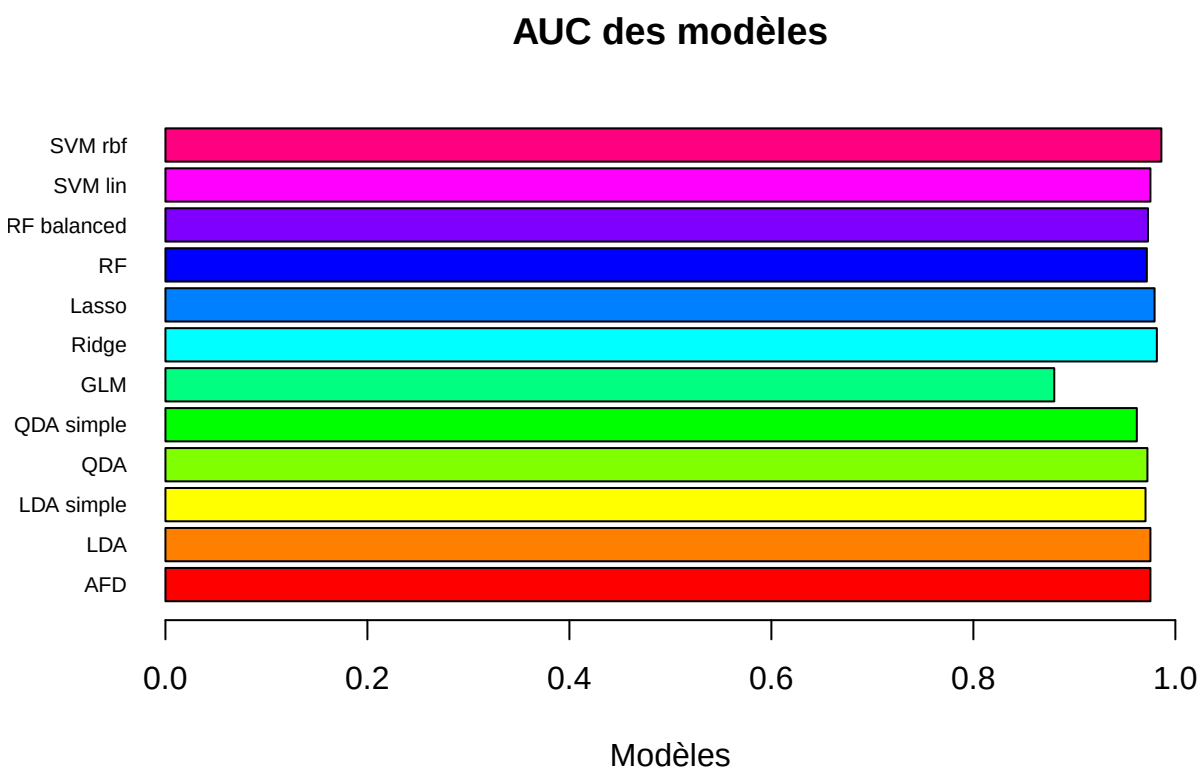


AUC

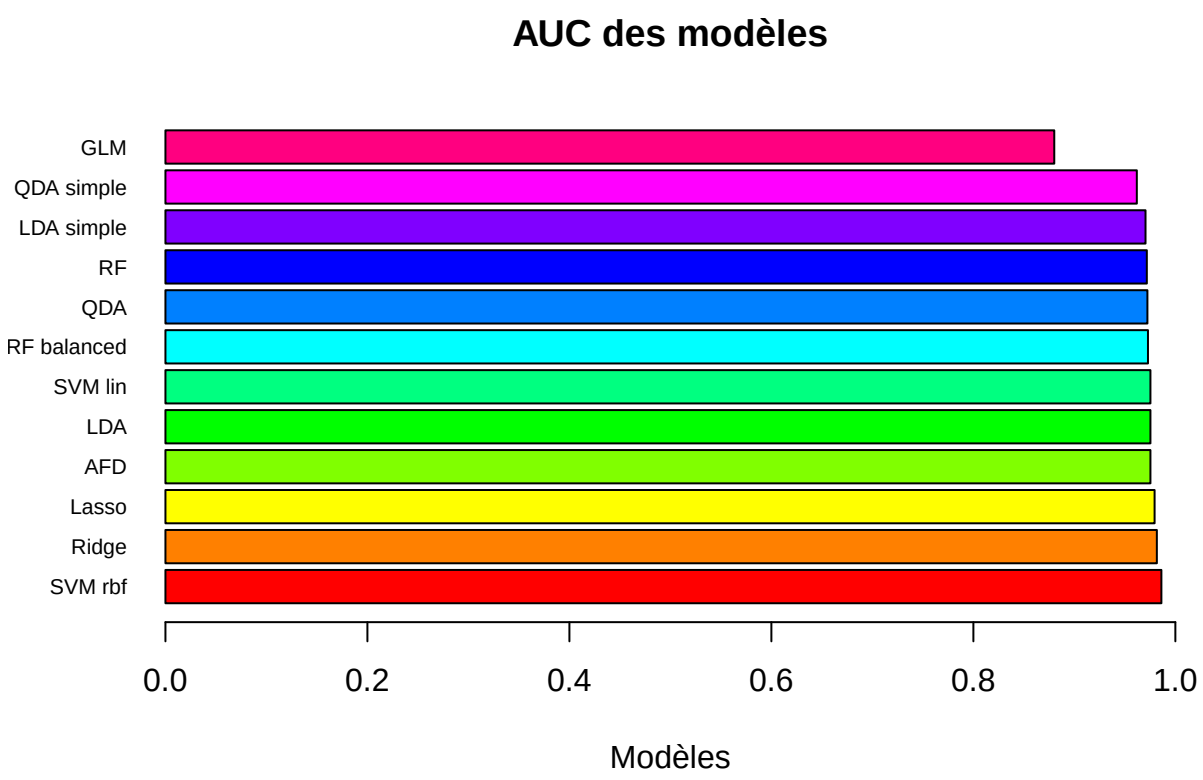
```
df.auc <- data.frame(model = legends_list, auc = sapply(roc_list, function(x) auc(x)))
df.auc
```

```
##          model       auc
## 1          AFD 0.9753086
## 2          LDA 0.9753086
## 3   LDA simple 0.9704452
## 4          QDA 0.9723158
## 5   QDA simple 0.9618406
## 6          GLM 0.8800973
## 7        Ridge 0.9816685
## 8        Lasso 0.9794239
## 9           RF 0.9717546
## 10 RF balanced 0.9728769
## 11     SVM lin 0.9753086
## 12     SVM rbf 0.9861579
```

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




```r
# AUC sorted
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)), ma
```



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.

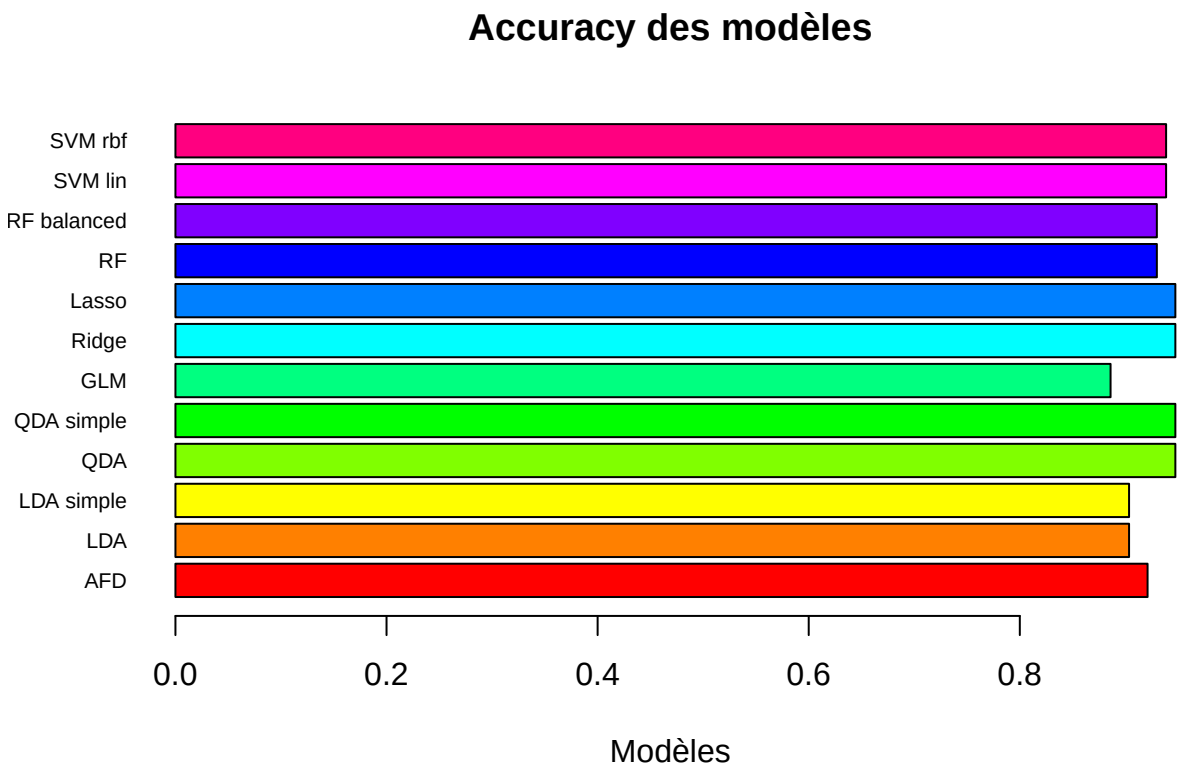
Comparaison de l'accuarcy des modèles

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

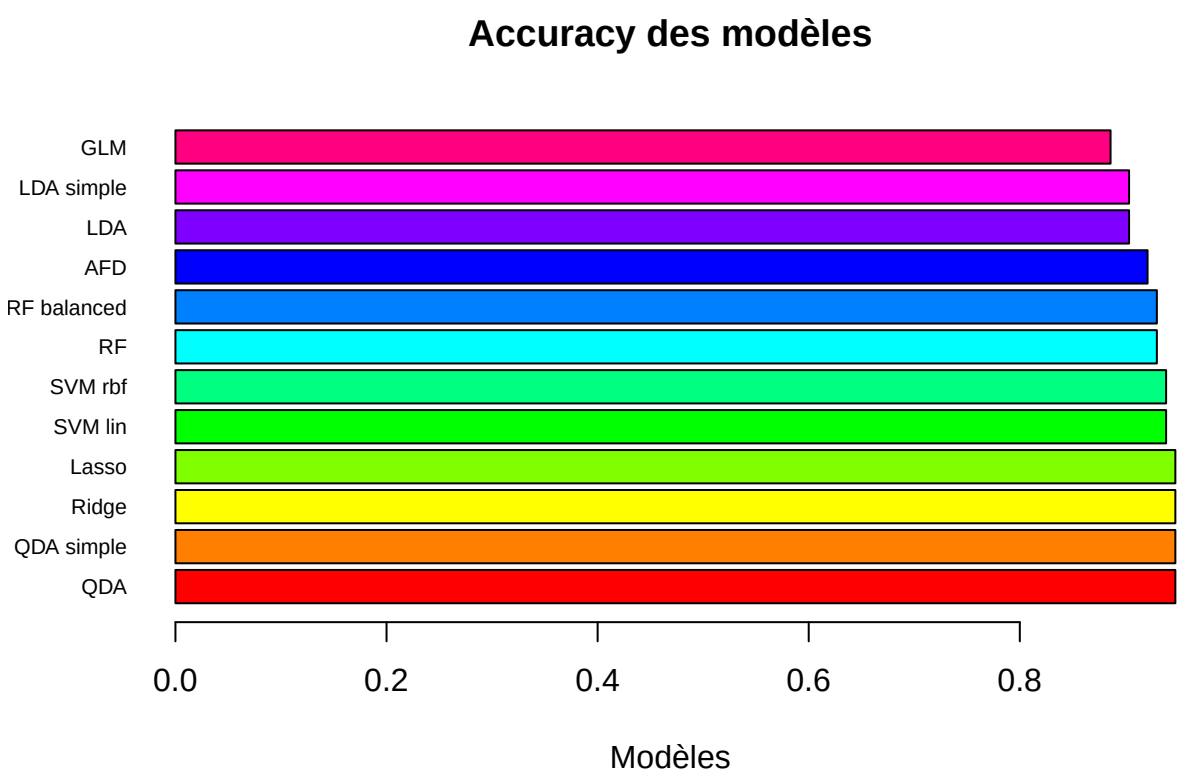
```
df_accuracy
```

```
##          model  accuracy
## 1          AFD 0.9210526
## 2          LDA 0.9035088
## 3   LDA simple 0.9035088
## 4          QDA 0.9473684
## 5   QDA simple 0.9473684
## 6          GLM 0.8859649
## 7        Ridge 0.9473684
## 8        Lasso 0.9473684
## 9           RF 0.9298246
## 10 RF balanced 0.9298246
## 11     SVM lin 0.9385965
## 12     SVM rbf 0.9385965
```

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




```r
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_accu
```



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.

Le modèle qui a le moins fonctionné est le modèle de régression logistique sans pénalisation. En effet, lorsque nous lancions l'analyse, nous obtenions des warnings ce qui signifiait que le modèle n'était pas bien calibré.

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 est le moins bon modèle que l'on a trouvé. En effet, il n'était pas bien calibré.

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.