Classification_supervisée

Tout d'abord, on filtre les lignes du dataframe pour ne conserver que celles où la variable depressif prend les valeurs "Oui" ou "Non". Ensuite, on convertit toutes les variables du dataframe en facteurs, ce qui permet de traiter chaque colonne comme une variable catégorielle. Enfin on conserve que les colonnes spécifiées : ville_foyer, mange_a_ta_faim, alcool, Conflits.Familiaux, tiers_temps, niveau_etude, anxiété, suivi_psycho, diagnostic_episode_depressif, depressif et suivi_medicale_depressif. Ces variables ont été sélectionné après l'analyse du chisq2.

```
sub_sante_mentale=read.csv("sante_mentale_regrouped.csv")
sub_sante_mentale <- sub_sante_mentale[sub_sante_mentale$depressif %in% c("Oui", "Non"), ]

# Convertir toutes les variables du dataframe en facteurs
for (column in names(sub_sante_mentale)) {
  sub_sante_mentale[[column]] <- as.factor(sub_sante_mentale[[column]])
}

# Subsetting the dataframe to keep only rows where depressif is "Oui" or "Non"
#sub_sante_mentale

# Assuming your data frame is named 'data'
sub_sante_mentale <- sub_sante_mentale[c("ville_foyer", "mange_a_ta_faim", "alcool", "Conflits.Familiaux
                        "tiers_temps", "niveau_etude", "anxiété", "suivi_psycho",
                        "diagnostic_episode_depressif", "depressif", "suivi_medicale_depressif")]

#sub_sante_mentale
attach(sub_sante_mentale)
```

Ce code permet de diviser le jeu de données initial en deux parties : une pour l'entraînement du modèle (sub_sante_mentale.train) et une pour son évaluation (sub_sante_mentale.test).

```
# Préparation des données
set.seed(1) # Permet de pouvoir reproduire les mêmes résultats
n <- nrow(sub_sante_mentale)
test.ratio <- 0.2 # Le taux de la db qu'on va utilisé comme test
n.test <- round(n * test.ratio) # Permet de savoir le nombre d'observation pour les test
tr <- sample(1:n, n.test)
sub_sante_mentale.test <- sub_sante_mentale[tr, ]
sub_sante_mentale.train <- sub_sante_mentale[-tr, ]
```

```
# Examiner les valeurs uniques par groupe de 'depressif'
aggregate(sub_sante_mentale[], by=list(sub_sante_mentale$depressif), function(x) length(unique(x)))
```

```
##   Group.1 ville_foyer mange_a_ta_faim alcool Conflits.Familiaux tiers_temps
## 1     Non           8               2      6                  3           2
## 2     Oui           8               2      4                  3           2
##   niveau_etude anxiété suivi_psycho diagnostic_episode_depressif depressif
## 1            3       3           2                            3         1
## 2            2       3           2                            3         1
##   suivi_medicale_depressif
```

```
## 1                         2
## 2                         2
```

```
library(FactoMineR)
library(factoextra)
```

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

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```
library(klaR)
```

```
## Loading required package: MASS
```

```
library(MASS)
library(rpart)
library(rpart.plot)
library(pROC)
```

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

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

LDA

```
# Assuming all data issues are fixed
res_lda <- lda(depressif ~ ., data = sub_sante_mentale.train)
```

ROC lda

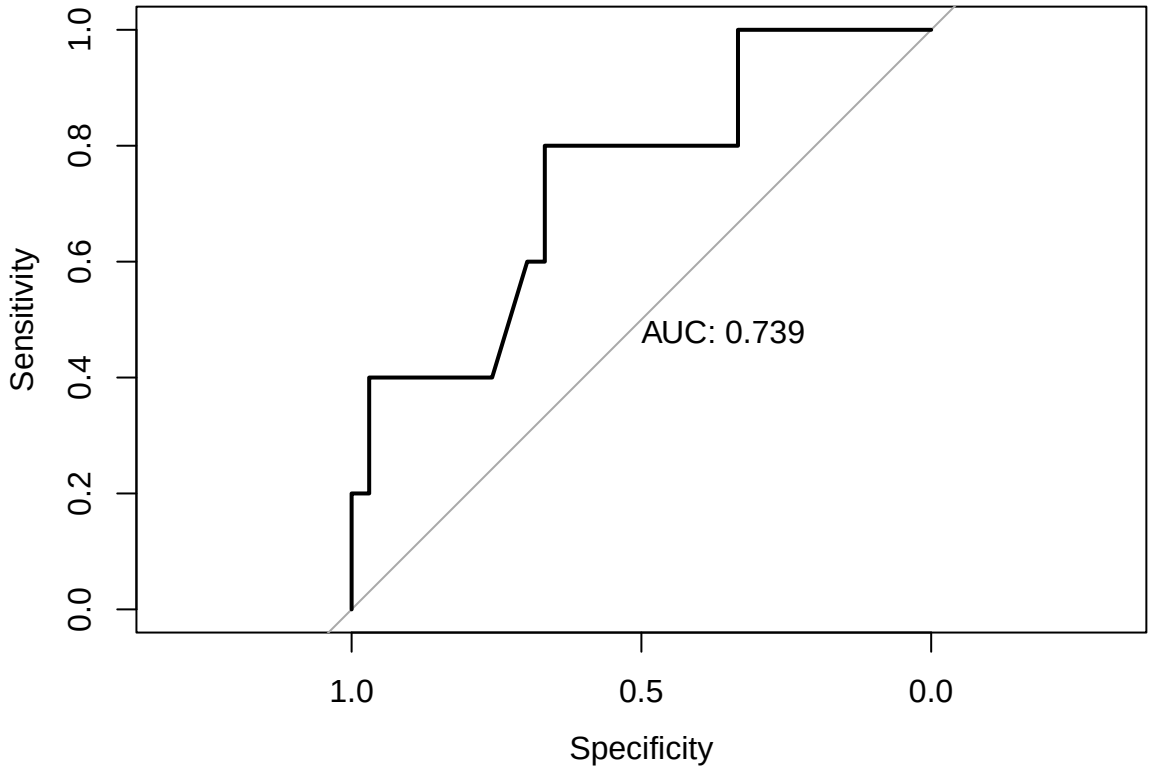
```
#proba a posteriori de succes (dans la deuxième colonne) :
pred_lda <- predict(res_lda,newdata=sub_sante_mentale.test)

ROC_lda <- roc(sub_sante_mentale.test$depressif, pred_lda$posterior[,2] )
```

```
## Setting levels: control = Non, case = Oui
```

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

```
plot(ROC_lda, print.auc=TRUE,  print.auc.y = 0.5)
```




```
ROC_lda$auc
```

```
## Area under the curve: 0.7394
```

Accuracy Lda

```
confusion_table_lda = table(Prédiction = pred_lda$class, Réalité = sub_sante_mentale.test$depressif)
confusion_table_lda
```

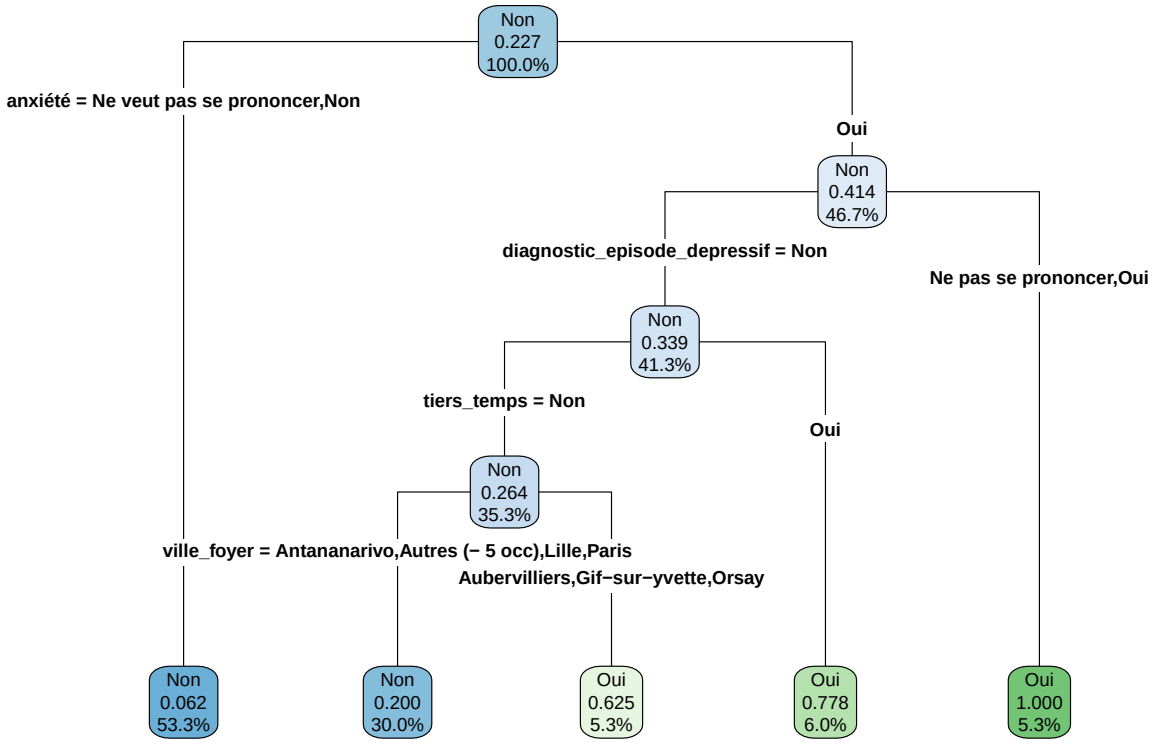
```
##           Réalité
## Prédiction Non Oui
##        Non  29   3
##        Oui   4   2
```

```
accuracy_lda = mean(pred_lda$class == sub_sante_mentale.test$depressif)
accuracy_lda
```

```
## [1] 0.8157895
```

CART

```
library(rpart)
arbre.opt <- rpart(depressif~., data=sub_sante_mentale.train)
rpart.plot(arbre.opt, type=4, digits=3, roundint=FALSE)
```



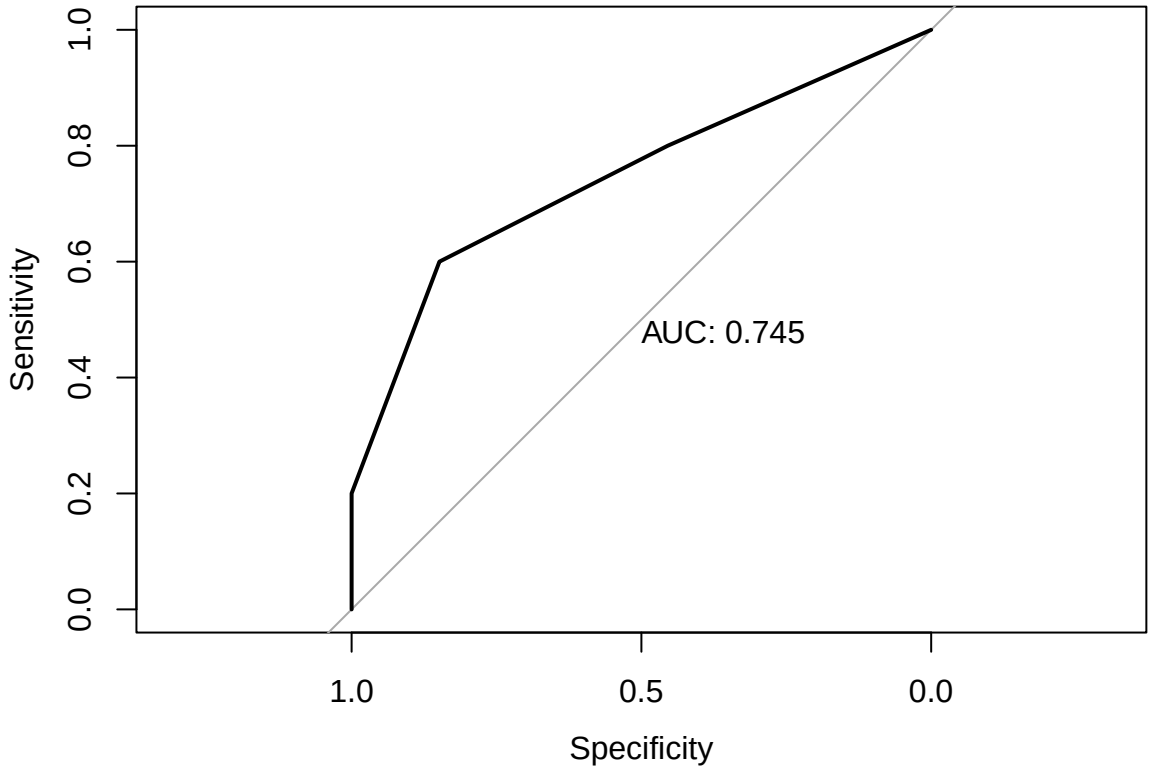
ROC Cart

```
# Courbe ROC pour CART
pred_cart <- predict(arbre.opt, sub_sante_mentale.test, type="prob")[,2]
ROC_cart <- roc(sub_sante_mentale.test$depressif, pred_cart)
```

```
## Setting levels: control = Non, case = Oui
```

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

```
plot(ROC_cart, print.auc=TRUE)
```



Accuracy Cart

```
pred_cart_class = predict(arbre.opt, sub_sante_mentale.test, type="class")
confusion_table_cart = table(Prédiction = pred_cart_class, Réalité = sub_sante_mentale.test$depressif)
confusion_table_cart
```

```
##           Réalité
## Prédiction Non Oui
##        Non  28   2
##        Oui   5   3
```

```
accuracy_cart = mean(pred_cart_class == sub_sante_mentale.test$depressif)
accuracy_cart
```

```
## [1] 0.8157895
```

Random Forest

```
library(randomForest)
```

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

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

```
##
## Attaching package: 'randomForest'
```

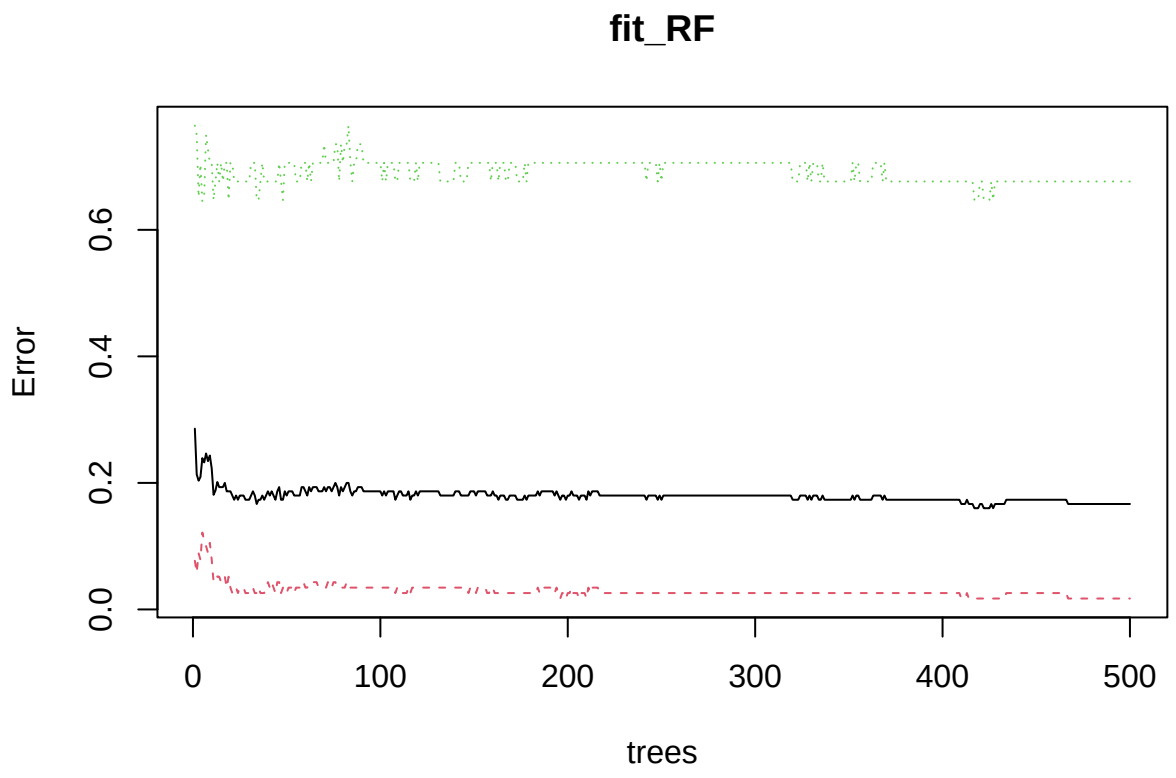
```
## The following object is masked from 'package:ggplot2':
```

```
##
##     margin
```

```
fit_RF <- randomForest(depressif~., data=sub_sante_mentale.train)
fit_RF
```

```
##
## Call:
##  randomForest(formula = depressif ~ ., data = sub_sante_mentale.train)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 16.67%
## Confusion matrix:
##     Non Oui class.error
## Non 114   2  0.01724138
## Oui  23  11  0.67647059
```

```
plot(fit_RF)
```



C'est très écarté, nous allons donc les rassembler en utilisant la méthode suivante :

Ce code utilise SMOTE pour équilibrer les classes dans le jeu de données d'entraînement, et affiche la nouvelle répartition des classes pour vérifier l'efficacité de l'équilibrage

SMOTE (Synthetic Minority Over-sampling Technique) est une méthode utilisée pour traiter le déséquilibre des classes dans les jeux de données. Le déséquilibre des classes se produit lorsqu'une classe (la classe minoritaire) est sous-représentée par rapport aux autres classes (la classe majoritaire). Ce déséquilibre peut

entraîner des modèles biaisés qui performent mal sur la classe minoritaire.

```
library(DMwR)
```

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

```
## Loading required package: grid
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
data.train.balanced <- SMOTE(depressif~., sub_sante_mentale.train)
table(data.train.balanced$depressif)
```

```
##
## Non Oui
## 136 102
```

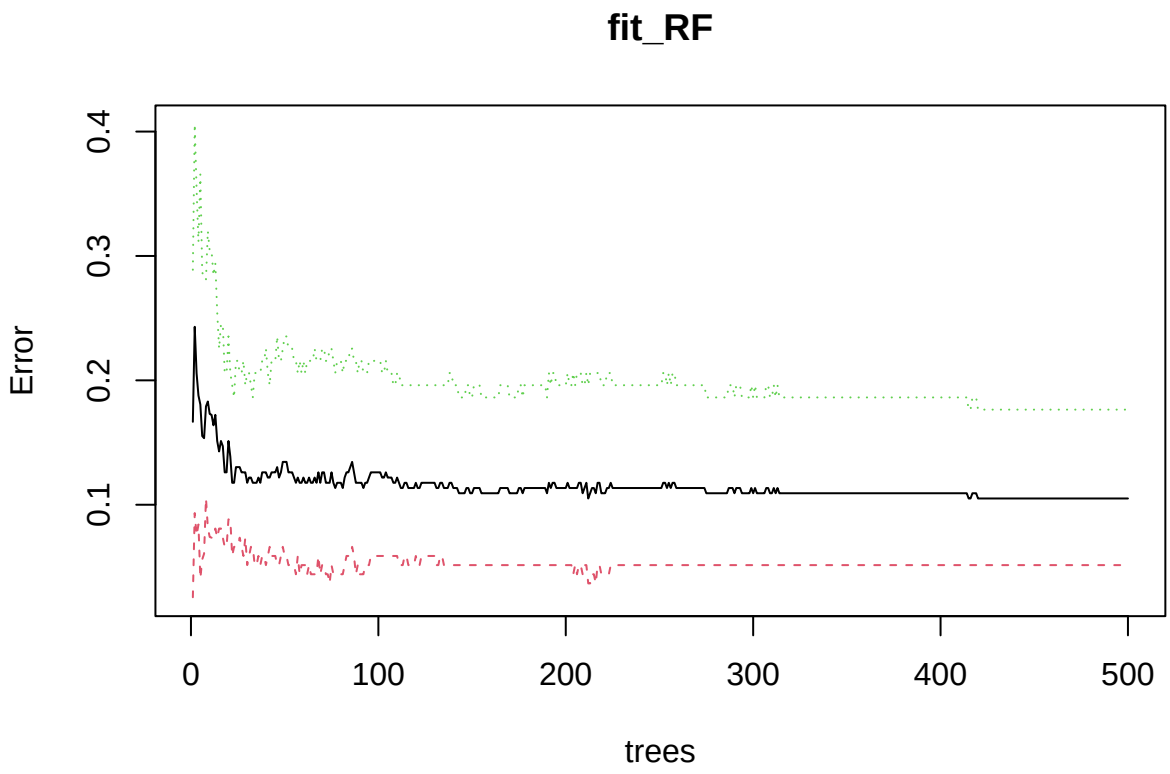
```
#data.train.balanced
```

Recalculons le tout avec la nouvelle base de donnée équilibrée par la méthode SMOTE

```
fit_RF <- randomForest(depressif~.,data.train.balanced)
fit_RF
```

```
##
## Call:
##  randomForest(formula = depressif ~ ., data = data.train.balanced)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 10.5%
## Confusion matrix:
##     Non Oui class.error
## Non 129   7  0.05147059
## Oui  18  84  0.17647059
```

```
plot(fit_RF)
```



Nous remarquons que c'est un peu mieux et que cela se rapproche plus.

ROC RandomForest

```
## aire sous courbe ROC
pred_RF = predict(fit_RF, sub_sante_mentale.test, type="prob")[,2]
ROC_RF <- roc(sub_sante_mentale.test$depressif, pred_RF)
```

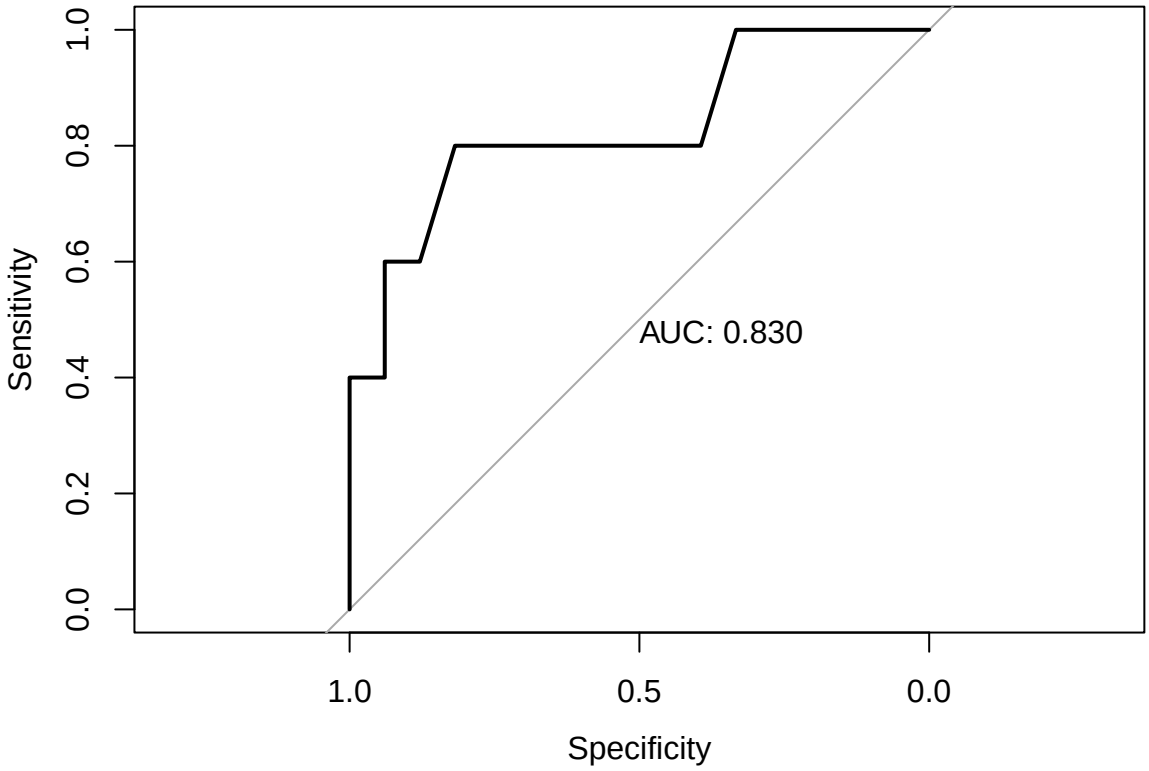
```
## Setting levels: control = Non, case = Oui
```

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

```
ROC_RF$auc
```

```
## Area under the curve: 0.8303
```

```
plot(ROC_RF, print.auc=TRUE,  print.auc.y = 0.5)
```



Nous remarquons que nous obtenons une meilleure accuracy et une meilleur AUC avec la méthode random forest que par rapport aux deux autres. (LDA, CART) Accuracy RandomForest

```
pred_RF_class = predict(fit_RF, sub_sante_mentale.test, type="class")
confusion_table_rf = table(Prédiction = pred_RF_class, Réalité = sub_sante_mentale.test$depressif)
confusion_table_rf
```

```
##           Réalité
## Prédiction Non Oui
##        Non  29   2
##        Oui   4   3
```

```
accuracy_rf = mean(pred_RF_class == sub_sante_mentale.test$depressif)
accuracy_rf
```

```
## [1] 0.8421053
```

AdaBoost

```
library(gbm)
```

```
## Loaded gbm 2.1.9
```

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

```
library(ROCR)  # To perform ROC analysis
```

Ici nous appliquons donc adaboost à la base de donnée. Nous faisons exprès de mettre en valeur numérique

pour adaboost

```
fit.adaboost=gbm(as.numeric(depressif)-1 ~., sub_sante_mentale.train, distribution = "adaboost")
# au lieu de réouvrir le jeu de données, on utilise depressif en tant que variable qualitative, mais on
# vous pouvez vérifier ce que fait :
  #  as.numeric(data.train$DIFF)
  #  as.numeric(data.train$DIFF) -1
fit.adaboost
```

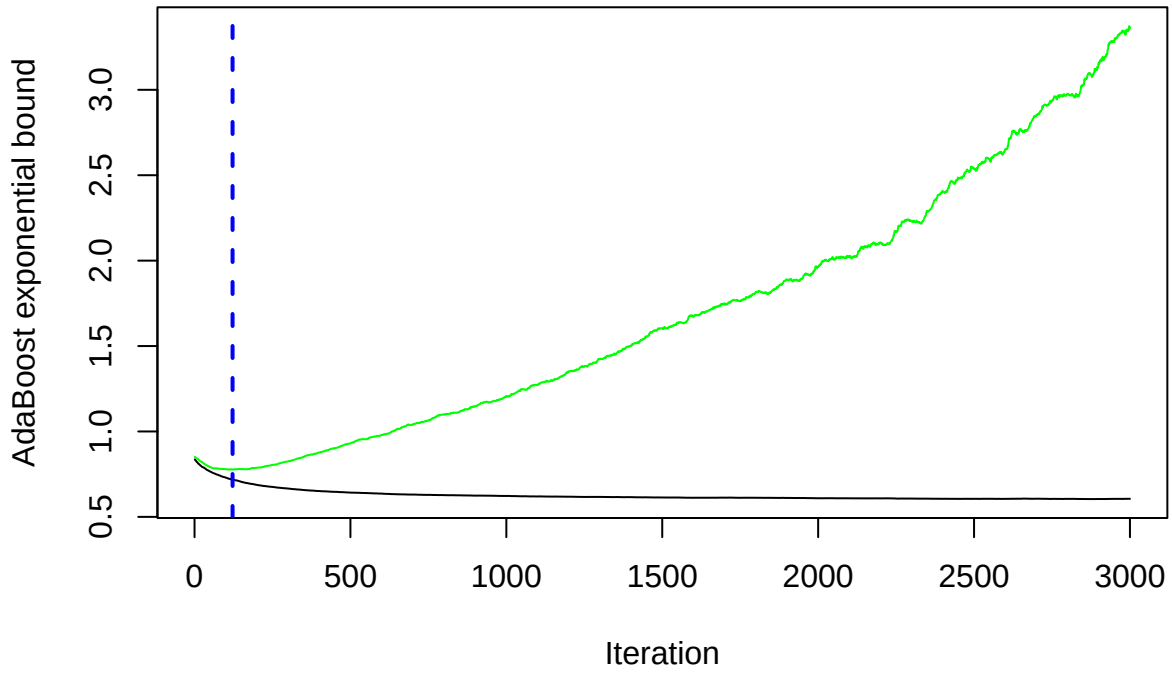
```
## gbm(formula = as.numeric(depressif) - 1 ~ ., distribution = "adaboost",
##     data = sub_sante_mentale.train)
## A gradient boosted model with adaboost loss function.
## 100 iterations were performed.
## There were 10 predictors of which 7 had non-zero influence.
```

Les commandes fournies ajustent un modèle de Gradient Boosting Machine (GBM) en utilisant l'algorithme AdaBoost sur le jeu de données sub_sante_mentale.train, avec la variable cible depressif convertie en format numérique. On ajuste donc le modèle en utilisant la validation croisée à 5 plis pour évaluer la performance sur différents nombres d'arbres (jusqu'à 3000) tout en appliquant un taux d'apprentissage (shrinkage) de 0,01. L'objectif de ce processus est d'optimiser le nombre d'arbres (B.opt) pour l'algorithme AdaBoost en identifiant le point où l'erreur de validation croisée est minimisée.

```
### Calibrer B=n.tree par cross-validation :
fit.adaboost=gbm(as.numeric(depressif)-1 ~., sub_sante_mentale.train, distribution = "adaboost",cv.folds
gbm.perf(fit.adaboost)
```

```
## [1] 122
```

```
B.opt = gbm.perf(fit.adaboost, method="cv")
```



La courbe montre que le modèle GBM atteint la meilleure performance sur l'ensemble de validation à environ 100 itérations. Au-delà de ce point, ajouter plus d'arbres conduit à un surapprentissage, comme l'indique l'augmentation de l'erreur de validation croisée.

Réappliquons alors l'adaboost en prenant en compte le nombre d'arbre B.opt

```
## prédiction :
pred_adaboost = predict(fit.adaboost, newdata=sub_sante_mentale.test, type = "response", n.trees = B.op

# Pour le transformer en class :
class_adaboost = 1 * (pred_adaboost > 1/2)
```

Accuracy Adaboost

```
confusion_table_adaboost = table(Prédiction = class_adaboost, Réalité = sub_sante_mentale.test$depressi
confusion_table_adaboost
```

```
##           Réalité
## Prédiction Non Oui
##          0  33   5
```

```
accuracy_adaboost = mean(class_adaboost == sub_sante_mentale.test$depressif)
accuracy_adaboost
```

```
## [1] 0
```

ROC Adaboost

```
## aire sous courbe ROC
ROC_adaboost <- roc(sub_sante_mentale.test$depressif, pred_adaboost)
```

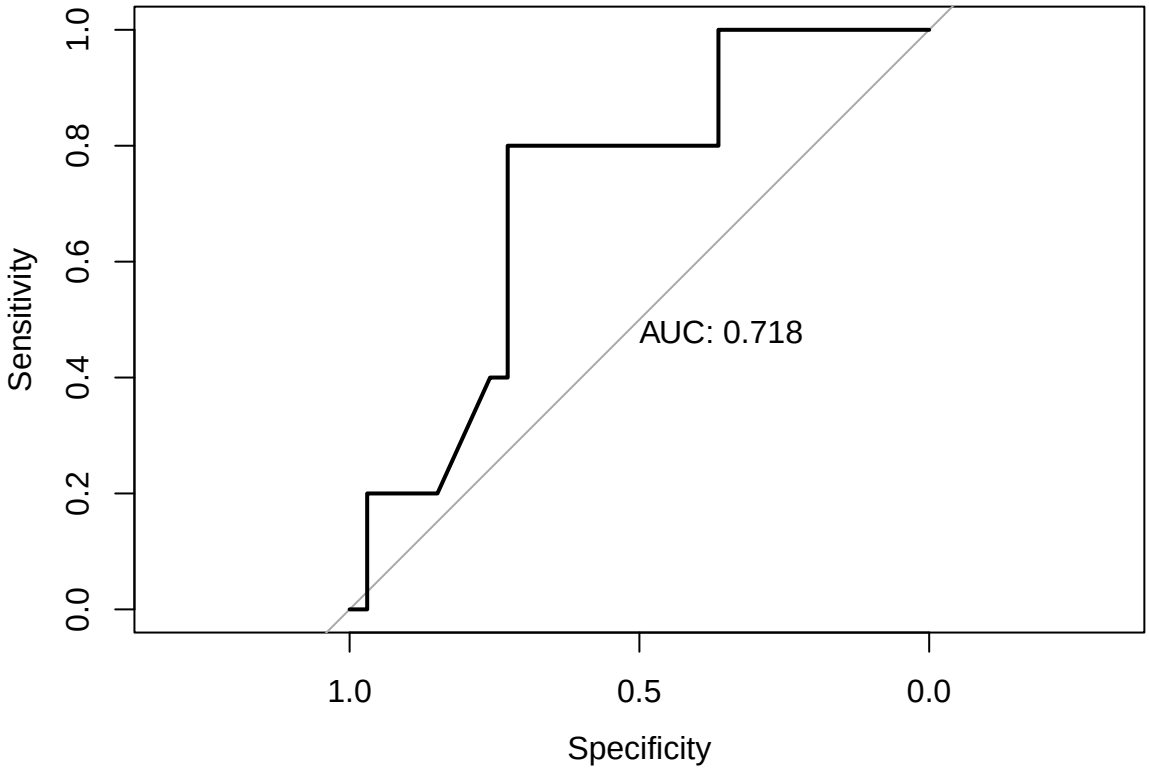
```
## Setting levels: control = Non, case = Oui
```

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

```
ROC_adaboost$auc
```

```
## Area under the curve: 0.7182
```

```
plot(ROC_adaboost, print.auc=TRUE,  print.auc.y = 0.5)
```



La courbe ROC montre que le modèle a une performance acceptable, avec une AUC de 0.779. Cela indique que le modèle est raisonnablement bon pour distinguer entre les classes positives et négatives, performe significativement mieux que le hasard mais n'atteint pas encore la catégorie de performance "bonne".

Comparaison des modèles

```
# Comparaison ----------------------------------------------------------------
result=matrix(NA, ncol=4, nrow=2)
rownames(result)=c('accuracy', 'AUC')
colnames(result)=c('lda', 'cart', 'RF', "adaboost")
result[1,]= c(accuracy_lda, accuracy_cart, accuracy_rf,accuracy_adaboost)
result[2,]=c(ROC_lda$auc, ROC_cart$auc, ROC_RF$auc,  ROC_adaboost$auc)
result
```

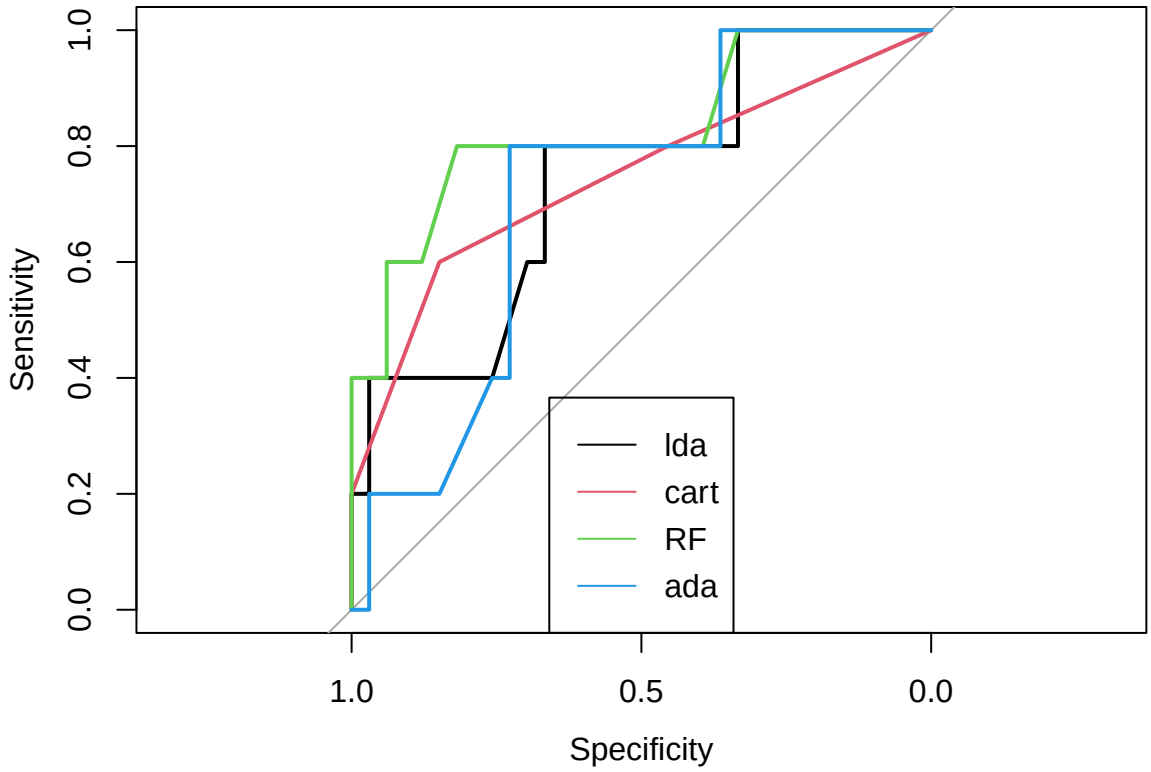
```
##                lda      cart        RF  adaboost
## accuracy 0.8157895 0.8157895 0.8421053 0.0000000
## AUC      0.7393939 0.7454545 0.8303030 0.7181818
```

```
apply(result,1, which.max )
```

```
## accuracy      AUC
##        3        3
```

```
plot(ROC_lda, xlim=c(1,0))
plot(ROC_cart, add=TRUE, col=2)
plot(ROC_RF, add=TRUE, col=3)
plot(ROC_adaboost, add=TRUE, col=4)
```

```r
legend('bottom', col=1:4, paste(c('lda', 'cart', 'RF', "ada")),  lwd=1)
```



Parmi les modèles comparés, le modèle de Random Forest (courbe verte) démontre la meilleure performance en termes de sensibilité et de spécificité, suivi par l'AdaBoost (courbe bleue) et l'analyse discriminante linéaire (LDA, courbe noire). La performance du modèle CART n'est pas clairement visible, mais elle est vraisemblablement moins efficace selon les courbes présentées. Cette analyse suggère que le modèle de Random Forest est le classificateur le plus efficace pour ce jeu de données afin de déterminer si quelqu'un est possiblement dépressif ou non .