Machine Learning Diabetes

Lansana CISSE

2024-03-16

```
# Definir le repertoire de travail
setwd("C:/Users/tandian/Desktop/tp")
```

```
# chargement des données
data <- read.csv("diabetes.csv", header = TRUE)
```

```
# Afficher la structure des données
str(data)
```

```
## 'data.frame':    768 obs. of  9 variables:
##  $ Pregnancies             : int  6 1 8 1 0 5 3 10 2 8 ...
##  $ Glucose                 : int  148 85 183 89 137 116 78 115 197 125 ...
##  $ BloodPressure           : int  72 66 64 66 40 74 50 0 70 96 ...
##  $ SkinThickness           : int  35 29 0 23 35 0 32 0 45 0 ...
##  $ Insulin                 : int  0 0 0 94 168 0 88 0 543 0 ...
##  $ BMI                     : num  33.6 26.6 23.3 28.1 43.1 25.6 31 35.3 30.5 0 ...
##  $ DiabetesPedigreeFunction: num  0.627 0.351 0.672 0.167 2.288 ...
##  $ Age                     : int  50 31 32 21 33 30 26 29 53 54 ...
##  $ Outcome                 : int  1 0 1 0 1 0 1 0 1 1 ...
```

1. Pretaitement des données

1.1 Identification des variables quantitatives et catégorielles

```
# Separer les variables quantitatives et catégorielles
var_cat <- data["Outcome"] # outcome est la seule variable catégorielle
var_quant <- data[-c(9)] # les autres variables sont quantitatives
```

```
# Convertir la variable Outcome en facteur
data$Outcome <- as.factor(data$Outcome)
```

```
# Afficher la struture des variables quantitatives
str(var_quant)
```

```
## 'data.frame':    768 obs. of  8 variables:
##  $ Pregnancies             : int  6 1 8 1 0 5 3 10 2 8 ...
##  $ Glucose                 : int  148 85 183 89 137 116 78 115 197 125 ...
```

```
##  $ BloodPressure           : int  72 66 64 66 40 74 50 0 70 96 ...
##  $ SkinThickness           : int  35 29 0 23 35 0 32 0 45 0 ...
##  $ Insulin                 : int  0 0 0 94 168 0 88 0 543 0 ...
##  $ BMI                     : num  33.6 26.6 23.3 28.1 43.1 25.6 31 35.3 30.5 0 ...
##  $ DiabetesPedigreeFunction: num  0.627 0.351 0.672 0.167 2.288 ...
##  $ Age                     : int  50 31 32 21 33 30 26 29 53 54 ...
```

```
# Afficher la struture des variables catégorielles
str(var_cat)
```

```
## 'data.frame':    768 obs. of  1 variable:
##  $ Outcome: int  1 0 1 0 1 0 1 0 1 1 ...
```

1.2 Identification et traitement des valeurs manquantes

A l'exception de la variable Outcome (variable cible) et de la variable pregnancies Nous allons remplacer les valeurs manquantes representées par 0 par NA pour faciliter le traitement des valeurs manquantes

```
# Remplacer les valeurs manquantes par NA
colonnes <- c("Glucose", "BloodPressure", "SkinThickness", "Insulin",
              "BMI", "DiabetesPedigreeFunction", "Age")
data[colonnes] <- lapply(data[colonnes], function(x) ifelse(x == 0, NA, x))
```

```
# Afficher le nombre de valeurs manquantes par variable
colSums(is.na(data))
```

```
##              Pregnancies                  Glucose            BloodPressure
##                        0                        5                       35
##            SkinThickness                  Insulin                      BMI
##                      227                      374                       11
## DiabetesPedigreeFunction                      Age                  Outcome
##                        0                        0                        0
```

1.3 Imputation des valeurs manquantes par regression lineaire

```
# Imputation des valeurs manquantes par regression lineaire
for (col in colnames(data)[2:8]) {
  data[[col]] <- ifelse(is.na(data[[col]]), predict(lm(data[[col]] ~
                                                       data$Outcome), data), data[[col]])
}
```

```
# Affichons les statistiques descriptives des variables quantitatives
summary(data[, -9]) # on exclut la variable Outcome
```

```
##   Pregnancies        Glucose       BloodPressure    SkinThickness
##  Min.   : 0.000   Min.   : 44.00   Min.   : 24.00   Min.   : 7.00
##  1st Qu.: 1.000   1st Qu.: 99.75   1st Qu.: 64.00   1st Qu.:25.00
##  Median : 3.000   Median :117.00   Median : 72.00   Median :28.00
##  Mean   : 3.845   Mean   :121.70   Mean   : 72.43   Mean   :29.25
```

```
##  3rd Qu.: 6.000   3rd Qu.:141.00   3rd Qu.: 80.00   3rd Qu.:33.00
##  Max.   :17.000   Max.   :199.00   Max.   :122.00   Max.   :99.00
##     Insulin           BMI        DiabetesPedigreeFunction      Age
##  Min.   : 14.0   Min.   :18.20   Min.   :0.0780           Min.   :21.00
##  1st Qu.:121.5   1st Qu.:27.50   1st Qu.:0.2437           1st Qu.:24.00
##  Median :130.3   Median :32.05   Median :0.3725           Median :29.00
##  Mean   :157.0   Mean   :32.45   Mean   :0.4719           Mean   :33.24
##  3rd Qu.:206.8   3rd Qu.:36.60   3rd Qu.:0.6262           3rd Qu.:41.00
##  Max.   :846.0   Max.   :67.10   Max.   :2.4200           Max.   :81.00
```

1.4 Separation des données en jeu d'apprentissage et jeu de test

```r
# fixer de la graine pour la reproductibilité
set.seed(1)
```

```r
# Partionner les données en jeu d'apprentissage et jeu de test

# Creer l'inidice de jeu de test
test_indices <- sample(1:nrow(data), 100)

# creer le jeu de test
test_set <- data[test_indices, ]

# creer le jeu d'apprentissage
train_set <- data[-test_indices, ]
```

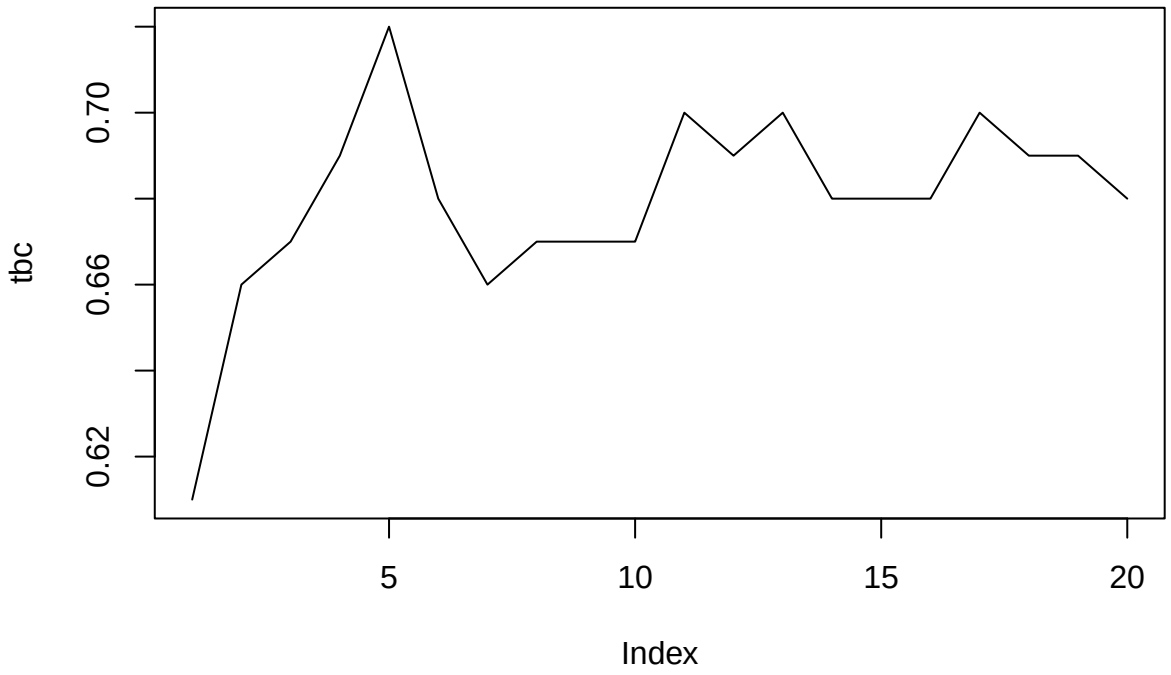
```r
# Separer les variables explicatives et la variable cible
train_x <- train_set[, -5]
train_y <- train_set$Outcome
test_x <- test_set[, -5]
test_y <- test_set$Outcome
```

2. Mise en oeuvre des algorithmes de Machine Learning

2.1 k-Nearest Neighbors (k-NN)

```r
library(class)

tbc=NULL
for (k in 1:20){
  knn_model <- knn(train = train_x, test = test_x, cl = train_y, k = k)
  tbc[k]=mean(knn_model==test_y)
}
plot(tbc,type='l')
```




```
# Determiner la valeur optimale de k
tcb <- which.max(tbc)
print(paste("La valeur optimale de k est", tcb))
```

```
## [1] "La valeur optimale de k est 5"
```

```
# creer le modele knn
model_knn <- knn(train = train_x, test = test_x, cl = train_y, k = tcb)
```

```
# Afficher la matrice de confusion
table(model_knn, test_y)
```

```
##          test_y
## model_knn  0  1
##         0 53 15
##         1 13 19
```

```
# Afficher l'accuracy
accurancy_knn = mean(model_knn == test_y)
accurancy_knn
```

```
## [1] 0.72
```

2.2 Random Forest

```
library(randomForest)
```

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

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

```
# creer le modele random forest
rf_model <- randomForest(Outcome ~ ., data = train_set, ntree = 100)

# predire la variable cible de l'ensemble de test
rf_model_pred <- predict(rf_model, test_set)
```

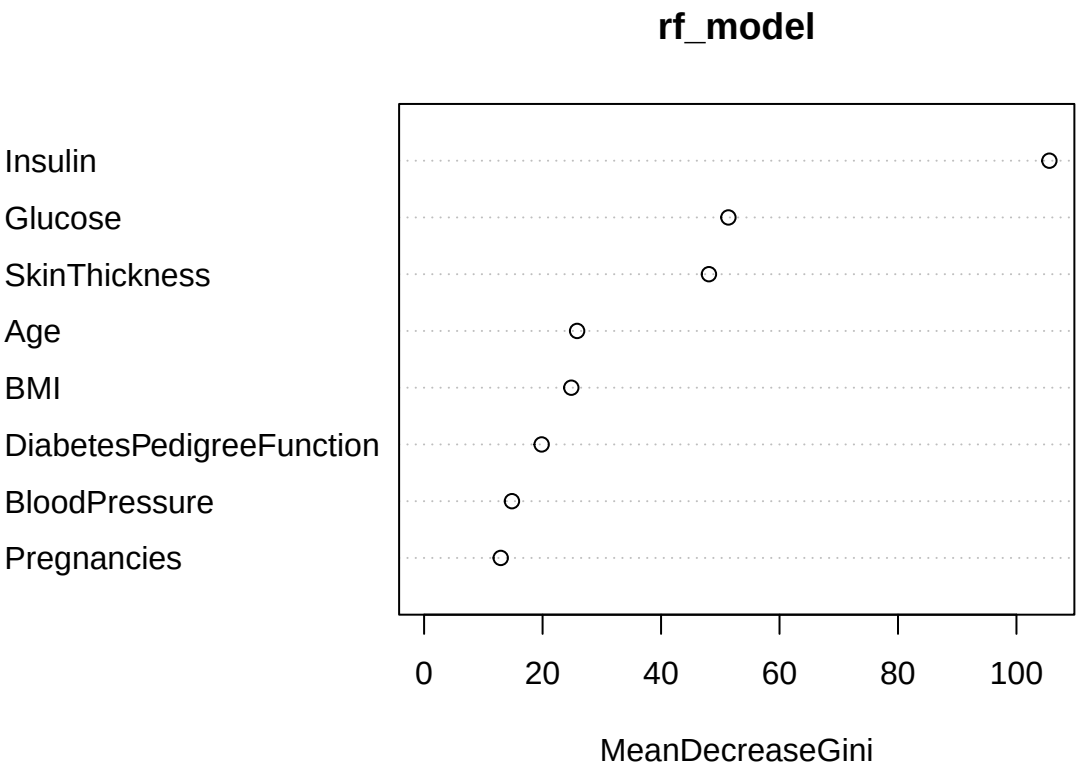
```
# Matrice de confusion
table(rf_model_pred, test_set$Outcome)
```

```
##
## rf_model_pred  0  1
##             0 60  9
##             1  6 25
```

```
# Calcul de l'accuracy
accurancy_rf = mean(rf_model_pred == test_set$Outcome)
accurancy_rf
```

```
## [1] 0.85
```

```
# Visualiser l'importance des variables
varImpPlot(rf_model)
```



2.3 Support Vector Machine (SVM)

```
library(e1071)

# creer le modele svm
svm_model <- svm(Outcome ~ ., data = train_set, kernel = "linear")

# predire la variable cible de l'ensemble de test
svm_model_pred <- predict(svm_model, test_set)

# afficher la matrice de confusion
table(svm_model_pred, test_set$Outcome)
```

```
##
## svm_model_pred  0  1
##              0 56 15
##              1 10 19
```

```
# Calcul de l'accuracy
accurancy_svm = mean(svm_model_pred == test_set$Outcome)
accurancy_svm
```

```
## [1] 0.75
```

2.4 Regression Logistique

```
# creer le modele de regression logistique
log_model <- glm(Outcome ~ ., data = train_set, family = binomial)

# predire la variable cible de l'ensemble de test
log_model_pred <- predict(log_model, test_set, type = "response")

# convertir les predictions en 0 et 1
log_model_pred <- ifelse(log_model_pred > 0.5, 1, 0)

# Afficher la matrice de confusion
table(log_model_pred, test_set$Outcome)
```

```
##
## log_model_pred  0  1
##              0 56 17
##              1 10 17
```

```
# affichage de l'accuracy
accurancy_log = mean(log_model_pred == test_set$Outcome)
accurancy_log
```

```
## [1] 0.73
```

2.5 Arbre de decision

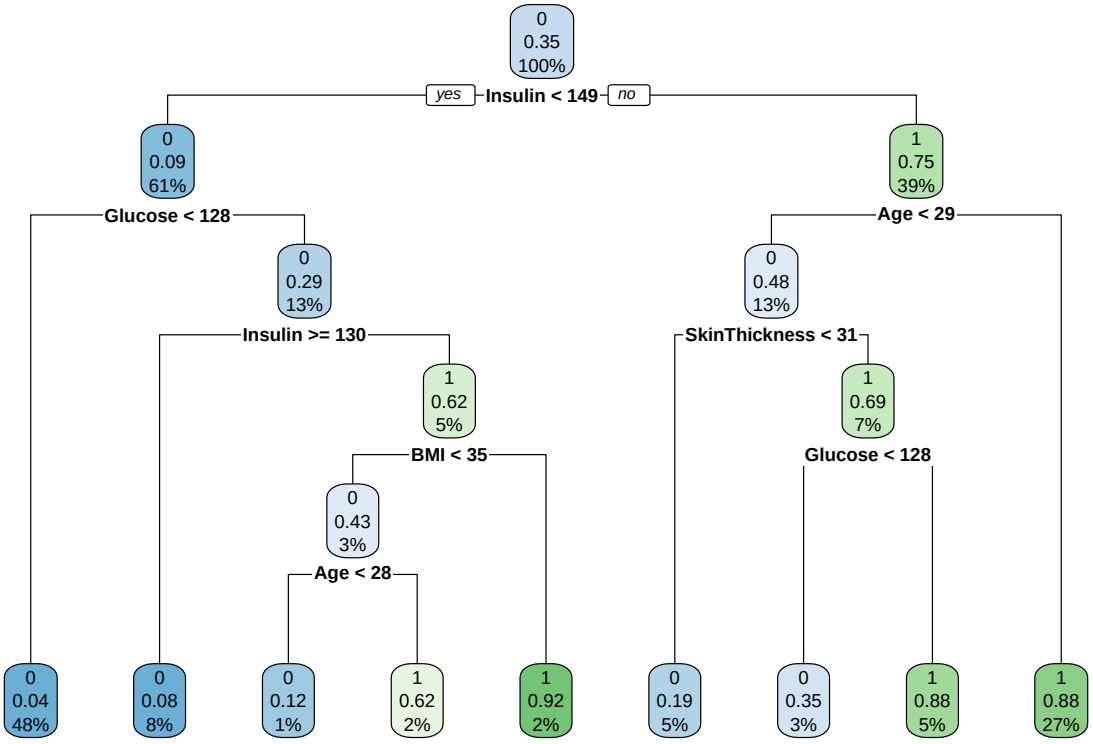
```
library(rpart)
library(rpart.plot)

# creer le dataframe pour le jeu d'apprentissage et le jeu de test
train_set <- as.data.frame(train_set)
test_set <- as.data.frame(test_set)

# # creer le modele de l'arbre de decision
tree_model <- rpart(Outcome ~ ., data = train_set, method = "class")

# predire la variable cible de l'ensemble de test
tree_model_pred <- predict(tree_model, test_set, type = "class")

# Afficher le graphique de l'arbre de decision
rpart.plot(tree_model)
```




```
# Afficher la matrice de confusion
table(tree_model_pred, test_set$Outcome)
```

```
##
## tree_model_pred  0  1
##               0 59 14
##               1  7 20
```

```r
# Calcul de l'accuracy
accurancy_tree = mean(tree_model_pred == test_set$Outcome)
accurancy_tree
```

```
## [1] 0.79
```

3. Conclusion : Comparaison des performances des algorithmes

```r
# Comparer les modeles en fonction de leur accuracy
accuracies <- c(mean(model_knn == test_y), mean(rf_model_pred == test_set$Outcome),
                mean(svm_model_pred == test_set$Outcome),
                mean(log_model_pred == test_set$Outcome),
                mean(tree_model_pred == test_set$Outcome))
names(accuracies) <- c("KNN", "Random Forest", "SVM", "Logistic Regression", "Decision Tree")

# Trier les accuracies par ordre décroissant
sorted_accuracies <- accuracies[order(-accuracies)]

# Afficher les accuracies triées
sorted_accuracies
```

```
##       Random Forest       Decision Tree                 SVM Logistic Regression
##                0.85                0.79                0.75                0.73
##                 KNN
##                0.72
```

```r
# Recuperer le meilleur modele
best_model <- names(sorted_accuracies)[1]

# Conclusion

cat("Le modèle", best_model, "est le meilleur modèle en terme de performance.\n")
```

```
## Le modèle Random Forest est le meilleur modèle en terme de performance.
```

```r
cat("Son accuracy est de", round(sorted_accuracies[1] * 100, 2), "%.\n")
```

```
## Son accuracy est de 85 %.
```

```r
# Afficher les variables déterminantes pour le meilleur modèle

if (best_model == "KNN") {
    print("Le modèle KNN ne permet pas de déterminer les variables les plus importantes.")
} else if (best_model == "Random Forest") {
    rf_variable_importance <- importance(rf_model)
    print("Variables déterminantes pour le modèle Random Forest :")
    print(rf_variable_importance)
} else if (best_model == "SVM") {
```

```
    svm_coeffs <- coef(svm_model)
    print("Variables déterminantes pour le modèle SVM :")
    print(svm_coeffs)
} else if (best_model == "Logistic Regression") {
    log_coefficients <- coef(log_model)
    print("Variables déterminantes pour le modèle de régression logistique :")
    print(log_coefficients)
} else if (best_model == "Decision Tree") {
    tree_variable_importance <- importance(tree_model)
    print("Variables déterminantes pour le modèle de l'arbre de décision :")
    print(tree_variable_importance)
}
```

```
## [1] "Variables déterminantes pour le modèle Random Forest :"
##                           MeanDecreaseGini
## Pregnancies                        12.93079
## Glucose                            51.36968
## BloodPressure                      14.82796
## SkinThickness                      48.08213
## Insulin                           105.55574
## BMI                                24.84920
## DiabetesPedigreeFunction           19.84177
## Age                                25.83125
```