Rain Australia

Notre jeu de données présente des relevés météorologiques en fonction de diverses villes en Australie.

Dans un premier temps nous ferons des analyses préliminaires afin d'en sortir des informations pertinentes qui nous permettrons de construire au mieux notre jeu de données.

La variable cible étant "RainTomorrow", nous lancerons plusieurs algorithmes de machine learning afin de prédire s'il pleuvra ou non le lendemain. Nous estimerons notre prédiction avec une matrice de confusion et divers estimateurs.

Pour information : RainToday / RainTomorrow = Yes si Rainfall > 1.

Import des libraries

```
knitr::opts_chunk$set(message = FALSE)
library(readxl)
library(tidyverse)
```

```
## Warning: le package 'tidyverse' a été compilé avec la version R 4.1.2
```

```
## -- Attaching packages --------------------------------------- tidyverse
1.3.1 --
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.1.0     v forcats 0.5.1
```

```
## Warning: le package 'tibble' a été compilé avec la version R 4.1.2
```

```
## Warning: le package 'tidyr' a été compilé avec la version R 4.1.2
```

```
## Warning: le package 'readr' a été compilé avec la version R 4.1.2
```

```
## Warning: le package 'dplyr' a été compilé avec la version R 4.1.1
```

```
## Warning: le package 'stringr' a été compilé avec la version R 4.1.1
```

```
## -- Conflicts ------------------------------------------
tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(PCAmixdata)
```

```
## Warning: le package 'PCAmixdata' a été compilé avec la version R 4.1.2

library(ggcorrplot)

## Warning: le package 'ggcorrplot' a été compilé avec la version R 4.1.2

library(ggplot2)
library(glmnet)

## Warning: le package 'glmnet' a été compilé avec la version R 4.1.2

## Le chargement a nécessité le package : Matrix

##
## Attachement du package : 'Matrix'

## Les objets suivants sont masqués depuis 'package:tidyr':
##
##     expand, pack, unpack

## Loaded glmnet 4.1-3

library(caret)

## Warning: le package 'caret' a été compilé avec la version R 4.1.1

## Le chargement a nécessité le package : lattice

##
## Attachement du package : 'caret'

## L'objet suivant est masqué depuis 'package:purrr':
##
##     lift

library(tidytable)

## Warning: le package 'tidytable' a été compilé avec la version R 4.1.2

##
## Attachement du package : 'tidytable'

## L'objet suivant est masqué depuis 'package:stats':
##
##     dt

library(e1071)

## Warning: le package 'e1071' a été compilé avec la version R 4.1.2

library(neuralnet)

## Warning: le package 'neuralnet' a été compilé avec la version R 4.1.2
```

```
##
## Attachement du package : 'neuralnet'

## L'objet suivant est masqué depuis 'package:dplyr':
##
##     compute

library(pROC)

## Warning: le package 'pROC' a été compilé avec la version R 4.1.2

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

##
## Attachement du package : 'pROC'

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

Chargement et visualisation des données

```
aus <- read.csv("weatherAUS.csv", header = TRUE, stringsAsFactors = TRUE)
head(aus)

##         Date Location MinTemp MaxTemp Rainfall Evaporation Sunshine
WindGustDir
## 1 2008-12-01   Albury    13.4    22.9      0.6          NA       NA
W
## 2 2008-12-02   Albury     7.4    25.1      0.0          NA       NA
WNW
## 3 2008-12-03   Albury    12.9    25.7      0.0          NA       NA
WSW
## 4 2008-12-04   Albury     9.2    28.0      0.0          NA       NA
NE
## 5 2008-12-05   Albury    17.5    32.3      1.0          NA       NA
W
## 6 2008-12-06   Albury    14.6    29.7      0.2          NA       NA
WNW
##   WindGustSpeed WindDir9am WindDir3pm WindSpeed9am WindSpeed3pm
Humidity9am
## 1            44          W        WNW           20           24
71
## 2            44        NNW        WSW            4           22
44
## 3            46          W        WSW           19           26
38
## 4            24         SE          E           11            9
45
```

```
## 5            41        ENE         NW            7           20
82
## 6            56          W          W           19           24
55
##   Humidity3pm Pressure9am Pressure3pm Cloud9am Cloud3pm Temp9am Temp3pm
## 1          22      1007.7      1007.1        8       NA    16.9    21.8
## 2          25      1010.6      1007.8       NA       NA    17.2    24.3
## 3          30      1007.6      1008.7       NA        2    21.0    23.2
## 4          16      1017.6      1012.8       NA       NA    18.1    26.5
## 5          33      1010.8      1006.0        7        8    17.8    29.7
## 6          23      1009.2      1005.4       NA       NA    20.6    28.9
##   RainToday RainTomorrow
## 1        No           No
## 2        No           No
## 3        No           No
## 4        No           No
## 5        No           No
## 6        No           No
```

Résumé des données

```
summary(aus)

##          Date            Location         MinTemp          MaxTemp
##  2013-03-01:    49   Canberra:  3436   Min.   :-8.50   Min.   :-4.80
##  2013-03-02:    49   Sydney  :  3344   1st Qu.: 7.60   1st Qu.:17.90
##  2013-03-03:    49   Adelaide:  3193   Median :12.00   Median :22.60
##  2013-03-04:    49   Brisbane:  3193   Mean   :12.19   Mean   :23.22
##  2013-03-05:    49   Darwin  :  3193   3rd Qu.:16.90   3rd Qu.:28.20
##  2013-03-06:    49   Hobart  :  3193   Max.   :33.90   Max.   :48.10
##  (Other)   :145166   (Other) :125908   NA's   :1485    NA's   :1261
##     Rainfall        Evaporation        Sunshine      WindGustDir
##  Min.   :  0.000   Min.   :  0.00   Min.   : 0.00   W      : 9915
##  1st Qu.:  0.000   1st Qu.:  2.60   1st Qu.: 4.80   SE     : 9418
##  Median :  0.000   Median :  4.80   Median : 8.40   N      : 9313
##  Mean   :  2.361   Mean   :  5.47   Mean   : 7.61   SSE    : 9216
##  3rd Qu.:  0.800   3rd Qu.:  7.40   3rd Qu.:10.60   E      : 9181
##  Max.   :371.000   Max.   :145.00   Max.   :14.50   (Other):88091
##  NA's   :3261      NA's   :62790    NA's   :69835   NA's   :10326
##  WindGustSpeed      WindDir9am      WindDir3pm     WindSpeed9am
##  Min.   :  6.00   N      :11758   SE     :10838   Min.   :  0.00
##  1st Qu.: 31.00   SE     : 9287   W      :10110   1st Qu.:  7.00
##  Median : 39.00   E      : 9176   S      : 9926   Median : 13.00
##  Mean   : 40.03   SSE    : 9112   WSW    : 9518   Mean   : 14.04
##  3rd Qu.: 48.00   NW     : 8749   SSE    : 9399   3rd Qu.: 19.00
##  Max.   :135.00   (Other):86812   (Other):91441   Max.   :130.00
##  NA's   :10263    NA's   :10566   NA's   : 4228   NA's   :1767
```

```
##   WindSpeed3pm    Humidity9am      Humidity3pm      Pressure9am
##  Min.   : 0.00   Min.   :  0.00   Min.   :  0.00   Min.   : 980.5
##  1st Qu.:13.00   1st Qu.: 57.00   1st Qu.: 37.00   1st Qu.:1012.9
##  Median :19.00   Median : 70.00   Median : 52.00   Median :1017.6
##  Mean   :18.66   Mean   : 68.88   Mean   : 51.54   Mean   :1017.6
##  3rd Qu.:24.00   3rd Qu.: 83.00   3rd Qu.: 66.00   3rd Qu.:1022.4
##  Max.   :87.00   Max.   :100.00   Max.   :100.00   Max.   :1041.0
##  NA's   :3062    NA's   :2654     NA's   :4507     NA's   :15065
##   Pressure3pm        Cloud9am        Cloud3pm        Temp9am
##  Min.   : 977.1   Min.   :0.00    Min.   :0.00    Min.   :-7.20
##  1st Qu.:1010.4   1st Qu.:1.00    1st Qu.:2.00    1st Qu.:12.30
##  Median :1015.2   Median :5.00    Median :5.00    Median :16.70
##  Mean   :1015.3   Mean   :4.45    Mean   :4.51    Mean   :16.99
##  3rd Qu.:1020.0   3rd Qu.:7.00    3rd Qu.:7.00    3rd Qu.:21.60
##  Max.   :1039.6   Max.   :9.00    Max.   :9.00    Max.   :40.20
##  NA's   :15028    NA's   :55888   NA's   :59358   NA's   :1767
##     Temp3pm       RainToday     RainTomorrow
##  Min.   :-5.40   No  :110319   No  :110316
##  1st Qu.:16.60   Yes : 31880   Yes : 31877
##  Median :21.10   NA's:  3261   NA's:  3267
##  Mean   :21.68
##  3rd Qu.:26.40
##  Max.   :46.70
##  NA's   :3609
```

Trie du jeu de données par date

```
aus <- aus[order(aus$Date),]
```

Suppression des variables avec un % trop important de valeur “NA”

```
# Suppression des variables avec env. 50% de données manquantes. Remplacer
les NA par des moyennes ou médianes amènerait un effet aléatoire aux
variables. Nous perdrions de la pertinence sur notre jeu de données.
aus <- subset(aus, select = -c(Evaporation,Sunshine, Cloud3pm, Cloud9am))
```

Traitement des données manquantes

```
colSums(is.na(aus))
```

```
##          Date      Location       MinTemp       MaxTemp      Rainfall
##             0             0          1485          1261          3261
##   WindGustDir WindGustSpeed    WindDir9am    WindDir3pm  WindSpeed9am
##         10326         10263         10566          4228          1767
##  WindSpeed3pm   Humidity9am   Humidity3pm   Pressure9am   Pressure3pm
##          3062          2654          4507         15065         15028
##       Temp9am       Temp3pm     RainToday  RainTomorrow
##          1767          3609          3261          3267
```

```
# Suppression des dernières valeurs manquantes
aus <- drop_na(aus)

# Vérification que nous n'avons plus de NA
colSums(is.na(aus))
```

```
##          Date      Location       MinTemp       MaxTemp      Rainfall
##             0             0             0             0             0
##   WindGustDir WindGustSpeed    WindDir9am    WindDir3pm  WindSpeed9am
##             0             0             0             0             0
##  WindSpeed3pm   Humidity9am   Humidity3pm   Pressure9am   Pressure3pm
##             0             0             0             0             0
##       Temp9am       Temp3pm     RainToday  RainTomorrow
##             0             0             0             0
```

Description des données après nettoyage

```
summary(aus)
```

```
##          Date                    Location        MinTemp          MaxTemp
##  2013-09-11:    44   Darwin          : 3117   Min.   :-8.20   Min.   :
2.60
##  2013-10-16:    44   Hobart          : 3089   1st Qu.: 8.10   1st
Qu.:18.30
##  2013-10-23:    44   Perth           : 3037   Median :12.40   Median
:23.10
##  2013-10-24:    44   Brisbane        : 3020   Mean   :12.66   Mean
:23.66
##  2013-11-06:    44   MelbourneAirport: 2934   3rd Qu.:17.20   3rd
Qu.:28.70
##  2013-11-26:    44   SydneyAirport   : 2929   Max.   :33.90   Max.
:48.10
##  (Other)   :112661   (Other)         :94799
##     Rainfall        WindGustDir    WindGustSpeed      WindDir9am
##  Min.   :  0.000   W      : 8380   Min.   :  7.00   N      : 9815
##  1st Qu.:  0.000   SE     : 8087   1st Qu.: 31.00   SSE    : 8122
##  Median :  0.000   E      : 7835   Median : 39.00   E      : 7980
##  Mean   :  2.378   S      : 7728   Mean   : 40.79   SE     : 7824
##  3rd Qu.:  0.800   SSE    : 7728   3rd Qu.: 48.00   S      : 7516
```

```
##  Max.   :367.600   N      : 7679   Max.   :135.00   SW     : 6979
##                    (Other):65488                    (Other):64689
##    WindDir3pm     WindSpeed9am    WindSpeed3pm   Humidity9am
##  SE     : 8325   Min.   : 2.00   Min.   : 2.0   Min.   :  0.0
##  S      : 8136   1st Qu.: 9.00   1st Qu.:13.0   1st Qu.: 56.0
##  W      : 7948   Median :13.00   Median :19.0   Median : 68.0
##  SSE    : 7794   Mean   :15.18   Mean   :19.5   Mean   : 67.4
##  WSW    : 7772   3rd Qu.:20.00   3rd Qu.:24.0   3rd Qu.: 81.0
##  SW     : 7656   Max.   :87.00   Max.   :87.0   Max.   :100.0
##  (Other):65294
##   Humidity3pm      Pressure9am      Pressure3pm        Temp9am
##  Min.   :  0.00   Min.   : 980.5   Min.   : 977.1   Min.   :-3.10
##  1st Qu.: 36.00   1st Qu.:1012.8   1st Qu.:1010.3   1st Qu.:12.70
##  Median : 51.00   Median :1017.4   Median :1015.0   Median :17.10
##  Mean   : 50.67   Mean   :1017.4   Mean   :1015.0   Mean   :17.46
##  3rd Qu.: 65.00   3rd Qu.:1022.1   3rd Qu.:1019.7   3rd Qu.:22.00
##  Max.   :100.00   Max.   :1041.0   Max.   :1039.6   Max.   :40.20
##
##     Temp3pm      RainToday   RainTomorrow
##  Min.   : 1.70   No :87556   No :87906
##  1st Qu.:16.90   Yes:25369   Yes:25019
##  Median :21.60
##  Mean   :22.13
##  3rd Qu.:26.90
##  Max.   :46.70
##
```

Une fois les valeurs manquantes traitées, nous remarquons que nous sommes dans un contexte avec des données déséquilibrées. Nous pourrons de ce fait utiliser des techniques d'échantillonage qui nous permettront d'équilibrer celui-ci.

Gestion de la date

```
# Suppression de la date
aus <- subset(aus, select = -c(Date))

head(aus)
```

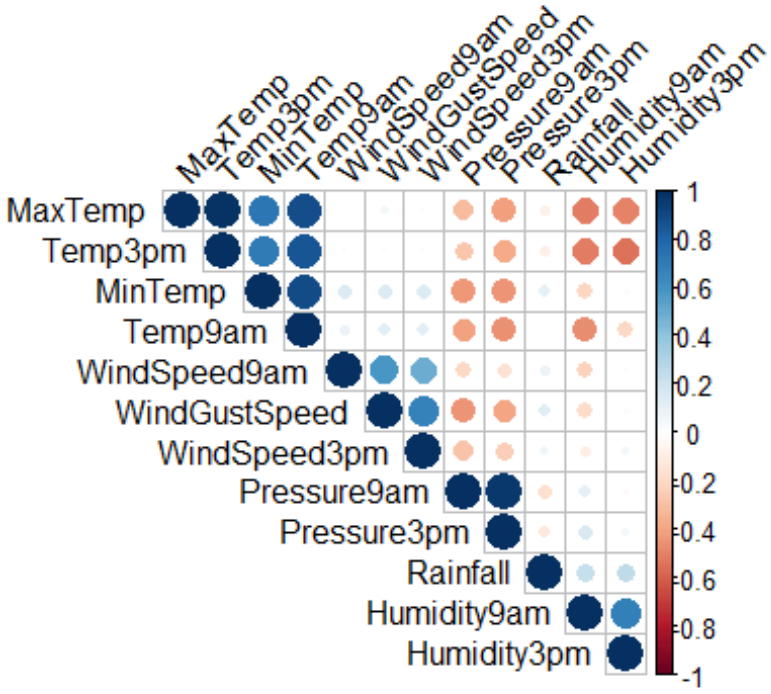
```
##   Location MinTemp MaxTemp Rainfall WindGustDir WindGustSpeed WindDir9am
## 1 Canberra     8.0    24.3      0.0          NW            30         SW
## 2 Canberra    14.0    26.9      3.6         ENE            39          E
## 3 Canberra    13.7    23.4      3.6          NW            85          N
## 4 Canberra    13.3    15.5     39.8          NW            54        WNW
## 5 Canberra     7.6    16.1      2.8         SSE            50        SSE
## 6 Canberra     6.2    16.9      0.0          SE            44         SE
##   WindDir3pm WindSpeed9am WindSpeed3pm Humidity9am Humidity3pm Pressure9am
## 1         NW            6           20          68          29      1019.7
```

```
## 2          W            4           17           80           36      1012.4
## 3        NNE            6            6           82           69      1009.5
## 4          W           30           24           62           56      1005.5
## 5        ESE           20           28           68           49      1018.3
## 6          E           20           24           70           57      1023.8
##   Pressure3pm Temp9am Temp3pm RainToday RainTomorrow
## 1      1015.0    14.4    23.6        No          Yes
## 2      1008.4    17.5    25.7       Yes          Yes
## 3      1007.2    15.4    20.2       Yes          Yes
## 4      1007.0    13.5    14.1       Yes          Yes
## 5      1018.5    11.1    15.4       Yes           No
## 6      1021.7    10.9    14.8        No           No
```

Quelques statistiques descriptives

Dans un premier temps nous effectuons une matrice de corrélation entre les diverses variables.

```
# Séparation du modèle :
X_aus <- subset(aus, select = -c(RainTomorrow))
y_aus <- aus$RainTomorrow

# Split quanti/quali
split <- splitmix(X_aus)

## Warning in splitmix(X_aus): Columns of class integer are considered as
## quantitative

#Matrices de corrélation
mcor <- cor(split$X.quanti)

# Corrélogramme avec corrplot
library(corrplot)

## Warning: le package 'corrplot' a été compilé avec la version R 4.1.1

corrplot(mcor, type="upper", order="hclust", tl.col="black", tl.srt=45)
```



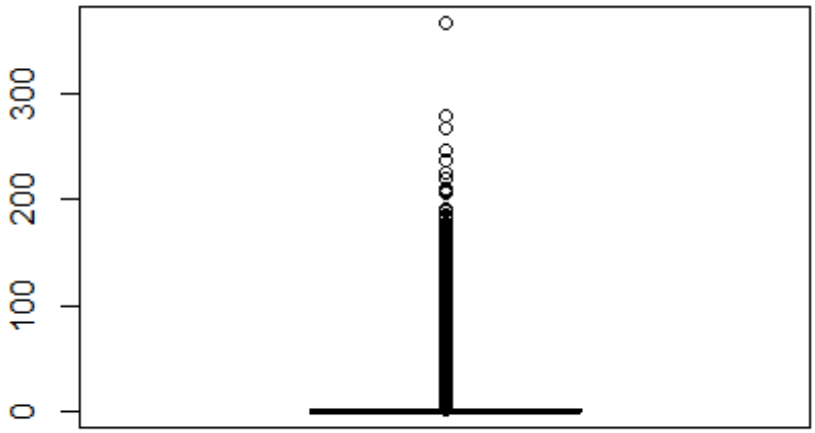
Avec le corrélogramme nous pouvons voir plus facilement que les variables liées à la température sont fortement corrélées. Les variables “Pressure9pm” et “Pressure3am” sont fortement liées aussi.

Il est évident de relever aussi que “MaxTemp”/“Temp3pm” et “MinTemp”/“Temp9am” ont une relation. Nous pouvons supprimer les variables “MaxTemp” et “MinTemp” qui vont amener la même information que les 2 autres.

```
# Suppression des variables MaxTemp, MinTemp
aus <- subset(aus, select = -c(MaxTemp, MinTemp))
```

Une question se pose sur la variable RainFall :

```
boxplot(aus$Rainfall)
```



Nous pouvons observer la faible visibilité de ce boxplot. Nous pouvons en tirer l'information suivante : en Australie, + de 75% du temps il ne pleut pas.

Centrage et reduction des données et préparation des données

```
"Jeux de données pour SVM"
```

```
## [1] "Jeux de données pour SVM"
```

```
# Séparation données quanti / quali
#split <- splitmix(aus)

# Centrage et réduction des données quanti
#aus_scaled <- as.data.frame(scale(split$X.quanti, center = T))

# Refonte du dataset pour avoir les quali et quanti
#aus <- cbind(split$X.quali, aus_scaled)

# Définition des X et y
#X <- subset(aus, select = -c(RainTomorrow))
#y <- aus$RainTomorrow

# Recodage des variables quali
#X <- get_dummies.(X, drop_first = TRUE)
```

```r
# X et X_nn définitif
#X <- subset(X, select = -c(Location, WindGustDir, WindDir9am, WindDir3pm,
RainToday))

"Jeu de données"
```

```
## [1] "Jeu de données"
```

```r
# Création dataset
aus <- aus[aus$Location == "AliceSprings" | aus$Location == "Brisbane" |
aus$Location == "Cairns" | aus$Location == "Perth" | aus$Location ==
"Sydney",]

# Séparation données quanti / quali
split <- splitmix(aus)
```

```
## Warning in splitmix(aus): Columns of class integer are considered as
## quantitative
```

```r
# Centrage et réduction des données quanti
aus_scaled <- as.data.frame(scale(split$X.quanti, center = T))

# Refonte du dataset pour avoir les quali et quanti
aus <- cbind(split$X.quali, aus_scaled)

# Définition des X et y
X <- subset(aus, select = -c(RainTomorrow))
y <- aus$RainTomorrow

# Recodage des variables quali
X <- get_dummies.(X, drop_first = TRUE)

# X et X_nn définitif
X <- subset(X, select = -c(Location, WindGustDir, WindDir9am, WindDir3pm,
RainToday))
```

Séparation des données

```r
# Split train/test
X_train <- as.matrix(X[1:9772,])
X_test <- as.matrix(X[9773:13959,])

y_train <- as.matrix(y[1:9772])
y_test <- as.matrix(y[9773:13959])
```

I) Support Vector Machines

a) SVM Linéaire

```
# Recodage des y_train
y_train_final <- ifelse(y_train == "Yes", 1, 0)

#Recodage y_test
y_test_final <- ifelse(y_test == "Yes", 1, 0)

# df train pour SVM
X_df <- data.frame(X_train)
#train_final <- X_df[,1:10]
#train_final <- cbind(X_df, X_df$RainToday_Yes)
train_final <-cbind(X_df, y_train_final)
colnames(train_final)[60:61] <- c("RainToday", "RainTomorrow")

# df test pour SVM
test_df <- data.frame(X_test)
test_final <- test_df[,1:60]
#test_final <- cbind(test_df, test_df$RainToday_Yes)
colnames(test_final)[60] <- "RainToday"

# Undersampling
#under_train_svm <- ovun.sample(RainTomorrow~., data=train_svm, p=0.5,
seed=5, method="under")$data

#Construction d'une solution pour tester plusieurs paramètres sans utiliser
la validation croisée
cost = c(0.1, 1)
epsilon = c(0.05, 0.5)

# Compteur
cpt = 1

# Résultats
acc_linear <- c()
precision <- c()
recall <- c()
f1_score <- c()

for(i in cost){
  for(j in epsilon){
    svm_fit_linear <- svm(formula = RainTomorrow~., data = train_final,
kernel = "linear", type = "C-classification", scale = FALSE, cost = i,
```

```
epsilon = j)

    # Prédiction
    pred_linear <- predict(svm_fit_linear, test_final, type = "class")

    # Matrice de confusion, hyper-paramètres et estimateurs
    mc_linear <-  confusionMatrix(pred_linear, factor(y_test_final), positive
= "1")

    # Affichage des données :
    cat("ALGORITHME N° :", cpt, "\n")
    cat("\n")
    cat("#################################\n")
    cat("\n")
    print(svm_fit_linear$call)
    cat("\n")
    cat("#################################\n")
    cat("\n")
    cat("Hyper-paramètres : \n")
    cat("Cost =", i,"\n")
    cat("Epsilon =", j, "\n")
    cat("\n")
    cat("#################################\n")
    cat("\n")
    cat("Matrice de confusion : \n")
    cat("\n")
    print(mc_linear$table)
    cat("\n")
    cat("#################################\n")
    cat("\n")
    cat("Accuracy :",round(mc_linear$overall[1]*100,2) ,"%\n")
    cat("Precision :",round(mc_linear$byClass[5]*100,2) ,"%\n")
    cat("Recall :",round(mc_linear$byClass[6]*100,2) ,"%\n")
    cat("F1 score :",round(mc_linear$byClass[7]*100,2) ,"%\n")
    cat("\n")
    cat("###############################################################\n")
    cat("\n")

    cpt = cpt + 1

    acc_linear <- rbind(acc_linear, round(mc_linear$overall[1]*100,2))
    precision <- rbind(precision, round(mc_linear$byClass[5]*100,2))
    recall <- rbind(recall, round(mc_linear$byClass[6]*100,2))
    f1_score <- rbind(f1_score, round(mc_linear$byClass[7]*100,2))
  }
}

## ALGORITHME N° : 1
##
## #################################
```

```
##
## svm(formula = RainTomorrow ~ ., data = train_final, kernel = "linear",
##     type = "C-classification", cost = i, epsilon = j, scale = FALSE)
##
## ###################################
##
## Hyper-paramètres :
## Cost = 0.1
## Epsilon = 0.05
##
## ###################################
##
## Matrice de confusion :
##
##           Reference
## Prediction    0    1
##          0 3146  477
##          1  134  430
##
## ###################################
##
## Accuracy : 85.41 %
## Precision : 76.24 %
## Recall : 47.41 %
## F1 score : 58.46 %
##
## ###################################################################
##
## ALGORITHME N° : 2
##
## ###################################
##
## svm(formula = RainTomorrow ~ ., data = train_final, kernel = "linear",
##     type = "C-classification", cost = i, epsilon = j, scale = FALSE)
##
## ###################################
##
## Hyper-paramètres :
## Cost = 0.1
## Epsilon = 0.5
##
## ###################################
##
## Matrice de confusion :
##
##           Reference
## Prediction    0    1
##          0 3146  477
##          1  134  430
##
```

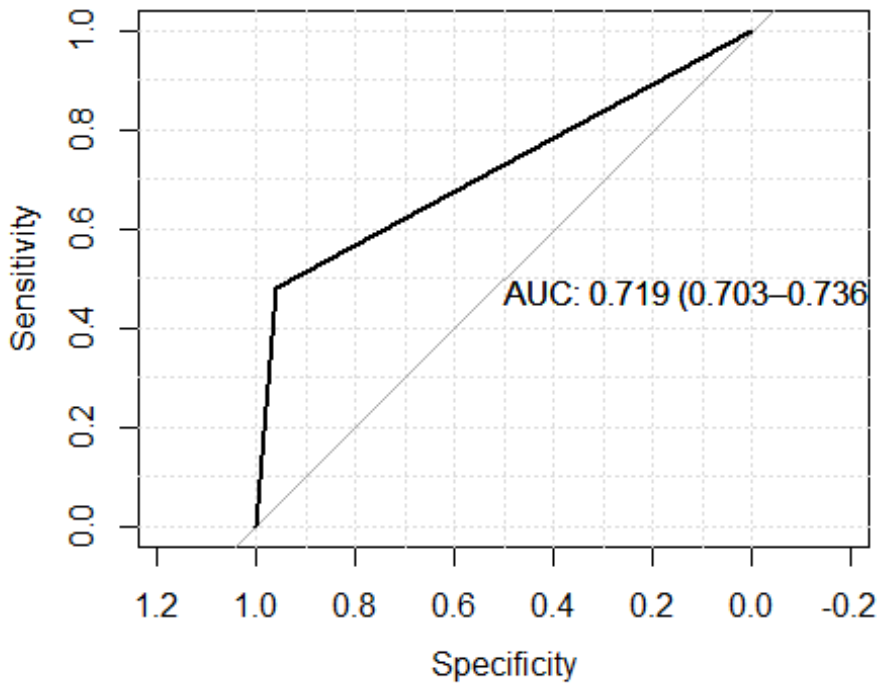
```
## ################################
##
## Accuracy : 85.41 %
## Precision : 76.24 %
## Recall : 47.41 %
## F1 score : 58.46 %
##
## ################################################################
##
## ALGORITHME N° : 3
##
## ################################
##
## svm(formula = RainTomorrow ~ ., data = train_final, kernel = "linear",
##     type = "C-classification", cost = i, epsilon = j, scale = FALSE)
##
## ################################
##
## Hyper-paramètres :
## Cost = 1
## Epsilon = 0.05
##
## ################################
##
## Matrice de confusion :
##
##           Reference
## Prediction    0    1
##          0 3142  471
##          1  138  436
##
## ################################
##
## Accuracy : 85.45 %
## Precision : 75.96 %
## Recall : 48.07 %
## F1 score : 58.88 %
##
## ################################################################
##
## ALGORITHME N° : 4
##
## ################################
##
## svm(formula = RainTomorrow ~ ., data = train_final, kernel = "linear",
##     type = "C-classification", cost = i, epsilon = j, scale = FALSE)
##
## ################################
##
## Hyper-paramètres :
```

```
## Cost = 1
## Epsilon = 0.5
##
## ################################
##
## Matrice de confusion :
##
##           Reference
## Prediction    0    1
##          0 3142  471
##          1  138  436
##
## ################################
##
## Accuracy : 85.45 %
## Precision : 75.96 %
## Recall : 48.07 %
## F1 score : 58.88 %
##
## ################################################################
```

Les meilleurs résultats sont cost = 1 et epsilon = 0.5 pour les SVM linéaires.

Courbe ROC pour les SVM linéaire :

```
svm_fit_linear_final <- svm(formula = RainTomorrow~., data = train_final,
kernel = "linear", type = "C-classification", scale = FALSE, cost = 1,
epsilon = 0.5)
pred_linear <- predict(svm_fit_linear_final, test_final, type = "class")

# Courbe ROC
pred_linear <- as.integer(pred_linear)
ROC_svm_linear <- roc(y_test_final,pred_linear,
            ci=TRUE, ci.alpha=0.9, stratified=FALSE,
            plot=TRUE, grid=TRUE,
            print.auc=TRUE)

## Warning in roc.default(y_test_final, pred_linear, ci = TRUE, ci.alpha =
0.9, :
## Deprecated use a matrix as response. Unexpected results may be produced,
please
## pass a vector or factor.
```




```
print(ROC_svm_linear)

##
## Call:
## roc.default(response = y_test_final, predictor = pred_linear,     ci =
TRUE, plot = TRUE, ci.alpha = 0.9, stratified = FALSE,     grid = TRUE,
print.auc = TRUE)
##
## Data: pred_linear in 3280 controls (y_test_final 0) < 907 cases
(y_test_final 1).
## Area under the curve: 0.7193
## 95% CI: 0.7027-0.7359 (DeLong)
```

Avec ces premiers résultats, nous observons que les hyper-paramètres du SVM linéaire changent peu les résultats.

b) SVM radial

```
# Ajout de l'hyper-paramètre gamma
gamma = c(0.1, 1)

# Réinitialisation compteur
cpt = 1
```

```r
# Résultats
acc_radial <- c()
precision <- c()
recall <- c()
f1_score <- c()

for(i in cost){
  for(j in gamma){
    # Fit avec un kernel radial
    svm_fit_rad <- svm(formula = RainTomorrow~., data = train_final, kernel = "radial", type = "C-classification", scale = FALSE, cost = i, gamma = j)

    # Prédiction
    pred_rad <- predict(svm_fit_rad, test_final, type = "class")

    # Matrice de confusion, hyper-paramètres et estimateurs
    mc_radial <-  confusionMatrix(pred_rad, factor(y_test_final), positive = "1")

    # Affichage des données :
    cat("ALGORITHME N° :", cpt, "\n")
    cat("\n")
    cat("##################################\n")
    cat("\n")
    print(svm_fit_rad$call)
    cat("\n")
    cat("##################################\n")
    cat("\n")
    cat("Hyper-paramètres : \n")
    cat("Cost =", i,"\n")
    cat("Gamma =", j, "\n")
    cat("\n")
    cat("##################################\n")
    cat("\n")
    cat("Matrice de confusion : \n")
    cat("\n")
    print(mc_radial$table)
    cat("\n")
    cat("##################################\n")
    cat("\n")
    cat("Accuracy : ",round(mc_radial$overall[1]*100,2) ,"%\n")
    cat("Precision : ",round(mc_radial$byClass[5]*100,2) ,"%\n")
    cat("Recall : ",round(mc_radial$byClass[6]*100,2) ,"%\n")
    cat("F1 score : ",round(mc_radial$byClass[7]*100,2) ,"%\n")
    cat("\n")
    cat("##################################################################\n")
    cat("\n")

    cpt = cpt + 1
```

```
    acc_radial <- rbind(acc_radial, round(mc_radial$overall[1]*100,2))
    precision <- rbind(acc_radial, round(mc_radial$byClass[5]*100,2))
    recall <- rbind(acc_radial, round(mc_radial$byClass[6]*100,2))
    f1_score <- rbind(acc_radial, round(mc_radial$byClass[7]*100,2))
  }
}
```

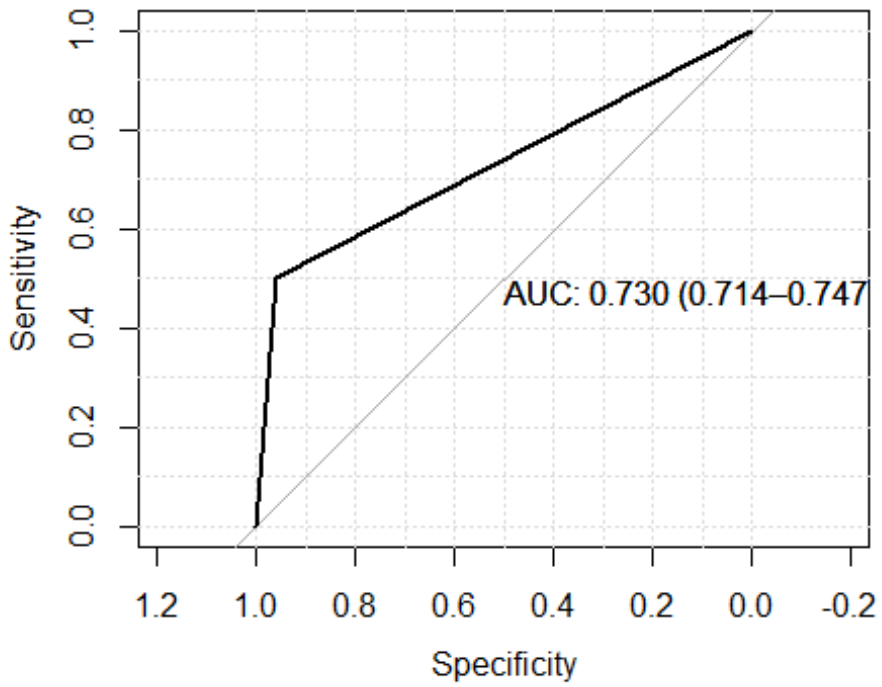
```
## ALGORITHME N° : 1
##
## ##################################
##
## svm(formula = RainTomorrow ~ ., data = train_final, kernel = "radial",
##     type = "C-classification", cost = i, gamma = j, scale = FALSE)
##
## ##################################
##
## Hyper-paramètres :
## Cost = 0.1
## Gamma = 0.1
##
## ##################################
##
## Matrice de confusion :
##
##           Reference
## Prediction    0    1
##          0 3180  512
##          1  100  395
##
## ##################################
##
## Accuracy :  85.38 %
## Precision :  79.8 %
## Recall :  43.55 %
## F1 score :  56.35 %
##
## ##################################################################
##
## ALGORITHME N° : 2
##
## ##################################
##
## svm(formula = RainTomorrow ~ ., data = train_final, kernel = "radial",
##     type = "C-classification", cost = i, gamma = j, scale = FALSE)
##
## ##################################
##
## Hyper-paramètres :
## Cost = 0.1
```

```
## Gamma = 1
##
## ################################
##
## Matrice de confusion :
##
##           Reference
## Prediction    0    1
##          0 3280  907
##          1    0    0
##
## ################################
##
## Accuracy :  78.34 %
## Precision :  NA %
## Recall :  0 %
## F1 score :  NA %
##
## ##################################################################
##
## ALGORITHME N° : 3
##
## ################################
##
## svm(formula = RainTomorrow ~ ., data = train_final, kernel = "radial",
##     type = "C-classification", cost = i, gamma = j, scale = FALSE)
##
## ################################
##
## Hyper-paramètres :
## Cost = 1
## Gamma = 0.1
##
## ################################
##
## Matrice de confusion :
##
##           Reference
## Prediction    0    1
##          0 3145  452
##          1  135  455
##
## ################################
##
## Accuracy :  85.98 %
## Precision :  77.12 %
## Recall :  50.17 %
## F1 score :  60.79 %
##
## ##################################################################
```

```
##
## ALGORITHME N° : 4
##
## ##################################
##
## svm(formula = RainTomorrow ~ ., data = train_final, kernel = "radial",
##     type = "C-classification", cost = i, gamma = j, scale = FALSE)
##
## ##################################
##
## Hyper-paramètres :
## Cost = 1
## Gamma = 1
##
## ##################################
##
## Matrice de confusion :
##
##           Reference
## Prediction    0    1
##          0 3267  886
##          1   13   21
##
## ##################################
##
## Accuracy :  78.53 %
## Precision :  61.76 %
## Recall :  2.32 %
## F1 score :  4.46 %
##
## ##################################################################
```

Courbe ROC pour SVM Radial :

```
svm_fit_radial_final <- svm(formula = RainTomorrow~., data = train_final,
kernel = "radial", type = "C-classification", scale = FALSE, cost = 1, gamma
= 0.1)
pred_radial <- predict(svm_fit_radial_final, test_final, type = "class")

# Courbe ROC
pred_radial <- as.integer(pred_radial)
ROC_svm_radial <- roc(y_test_final,pred_radial,
            ci=TRUE, ci.alpha=0.9, stratified=FALSE,
            plot=TRUE, grid=TRUE,
            print.auc=TRUE)

## Warning in roc.default(y_test_final, pred_radial, ci = TRUE, ci.alpha =
0.9, :
## Deprecated use a matrix as response. Unexpected results may be produced,
please
## pass a vector or factor.
```




```
print(ROC_svm_radial)

##
## Call:
## roc.default(response = y_test_final, predictor = pred_radial,     ci =
TRUE, plot = TRUE, ci.alpha = 0.9, stratified = FALSE,     grid = TRUE,
print.auc = TRUE)
##
## Data: pred_radial in 3280 controls (y_test_final 0) < 907 cases
(y_test_final 1).
## Area under the curve: 0.7302
## 95% CI: 0.7136-0.7469 (DeLong)
```

c) SVM polynomial

Puis avec un noyau polynomial de degré 2.

```
# Ajout de l'hyper-paramètre degree, coef0
degree = 2
coef0 = c(0.1, 1)

# Compteur
cpt = 1
```

```r
# Résultats
acc_poly <- c()
precision <- c()
recall <- c()
f1_score <- c()

for(i in cost){
  for(j in gamma){
    for(k in degree){
      for(l in coef0){
        # Fit avec un kernel linéaire
        svm_fit_poly <- svm(formula = RainTomorrow~., data = train_final,
kernel = "polynomial", type = "C-classification", scale = FALSE, cost = i,
gamma = j, degree = k, coef0 = l)

        # Prédiction
        pred_poly <- predict(svm_fit_poly, test_final, type = "class")

        # Matrice de confusion
        mc_poly <-  confusionMatrix(pred_poly, factor(y_test_final), positive
= "1")

        # Affichage des données :
        cat("ALGORITHME N° :", cpt, "\n")
        cat("\n")
        cat("#################################\n")
        cat("\n")
        print(svm_fit_poly$call)
        cat("\n")
        cat("#################################\n")
        cat("\n")
        cat("Hyper-paramètres : \n")
        cat("Cost =", i,"\n")
        cat("Gamma =", j, "\n")
        cat("Nombre de degrés =", k, "\n")
        cat("Coef0 =", l, "\n")
        cat("\n")
        cat("#################################\n")
        cat("\n")
        cat("Matrice de confusion : \n")
        cat("\n")
        print(mc_poly$table)
        cat("\n")
        cat("#################################\n")
        cat("\n")
        cat("Accuracy : ",round(mc_poly$overall[1]*100,2) ,"%\n")
        cat("Precision : ",round(mc_poly$byClass[5]*100,2) ,"%\n")
        cat("Recall : ",round(mc_poly$byClass[6]*100,2) ,"%\n")
        cat("F1 score : ",round(mc_poly$byClass[7]*100,2) ,"%\n")
        cat("\n")
```

```
cat("#################################################################\n")
        cat("\n")

        cpt = cpt + 1

        acc_poly <- rbind(acc_poly, round(mc_poly$overall[1]*100,2))
        precision <- rbind(acc_radial, round(mc_poly$byClass[5]*100,2))
        recall <- rbind(acc_radial, round(mc_poly$byClass[6]*100,2))
        f1_score <- rbind(acc_radial, round(mc_poly$byClass[7]*100,2))
      }
    }
  }
}
```

```
## ALGORITHME N° : 1
##
## ##################################
##
## svm(formula = RainTomorrow ~ ., data = train_final, kernel = "polynomial",
##     type = "C-classification", cost = i, gamma = j, degree = k,
##     coef0 = l, scale = FALSE)
##
## ##################################
##
## Hyper-paramètres :
## Cost = 0.1
## Gamma = 0.1
## Nombre de degrés = 2
## Coef0 = 0.1
##
## ##################################
##
## Matrice de confusion :
##
##           Reference
## Prediction    0    1
##          0 3180  517
##          1  100  390
##
## ##################################
##
## Accuracy :  85.26 %
## Precision :  79.59 %
## Recall :  43 %
## F1 score :  55.83 %
##
## ################################################################
##
## ALGORITHME N° : 2
```

```
##
## ###################################
##
## svm(formula = RainTomorrow ~ ., data = train_final, kernel = "polynomial",
##     type = "C-classification", cost = i, gamma = j, degree = k,
##     coef0 = l, scale = FALSE)
##
## ###################################
##
## Hyper-paramètres :
## Cost = 0.1
## Gamma = 0.1
## Nombre de degrés = 2
## Coef0 = 1
##
## ###################################
##
## Matrice de confusion :
##
##           Reference
## Prediction    0    1
##          0 3166  491
##          1  114  416
##
## ###################################
##
## Accuracy :  85.55 %
## Precision :  78.49 %
## Recall :  45.87 %
## F1 score :  57.9 %
##
## ####################################################################
##
## ALGORITHME N° : 3
##
## ###################################
##
## svm(formula = RainTomorrow ~ ., data = train_final, kernel = "polynomial",
##     type = "C-classification", cost = i, gamma = j, degree = k,
##     coef0 = l, scale = FALSE)
##
## ###################################
##
## Hyper-paramètres :
## Cost = 0.1
## Gamma = 1
## Nombre de degrés = 2
## Coef0 = 0.1
##
## ###################################
```

```
##
## Matrice de confusion :
##
##           Reference
## Prediction    0    1
##          0 3085  429
##          1  195  478
##
## ##################################
##
## Accuracy :  85.1 %
## Precision :  71.03 %
## Recall :  52.7 %
## F1 score :  60.51 %
##
## ####################################################################
##
## ALGORITHME N° : 4
##
## ##################################
##
## svm(formula = RainTomorrow ~ ., data = train_final, kernel = "polynomial",
##     type = "C-classification", cost = i, gamma = j, degree = k,
##     coef0 = l, scale = FALSE)
##
## ##################################
##
## Hyper-paramètres :
## Cost = 0.1
## Gamma = 1
## Nombre de degrés = 2
## Coef0 = 1
##
## ##################################
##
## Matrice de confusion :
##
##           Reference
## Prediction    0    1
##          0 3084  429
##          1  196  478
##
## ##################################
##
## Accuracy :  85.07 %
## Precision :  70.92 %
## Recall :  52.7 %
## F1 score :  60.47 %
##
## ####################################################################
```

```
##
## ALGORITHME N° : 5
##
## ################################
##
## svm(formula = RainTomorrow ~ ., data = train_final, kernel = "polynomial",
##     type = "C-classification", cost = i, gamma = j, degree = k,
##     coef0 = l, scale = FALSE)
##
## ################################
##
## Hyper-paramètres :
## Cost = 1
## Gamma = 0.1
## Nombre de degrés = 2
## Coef0 = 0.1
##
## ################################
##
## Matrice de confusion :
##
##           Reference
## Prediction    0    1
##          0 3141  457
##          1  139  450
##
## ################################
##
## Accuracy :  85.77 %
## Precision :  76.4 %
## Recall :  49.61 %
## F1 score :  60.16 %
##
## ################################################################
##
## ALGORITHME N° : 6
##
## ################################
##
## svm(formula = RainTomorrow ~ ., data = train_final, kernel = "polynomial",
##     type = "C-classification", cost = i, gamma = j, degree = k,
##     coef0 = l, scale = FALSE)
##
## ################################
##
## Hyper-paramètres :
## Cost = 1
## Gamma = 0.1
## Nombre de degrés = 2
## Coef0 = 1
```

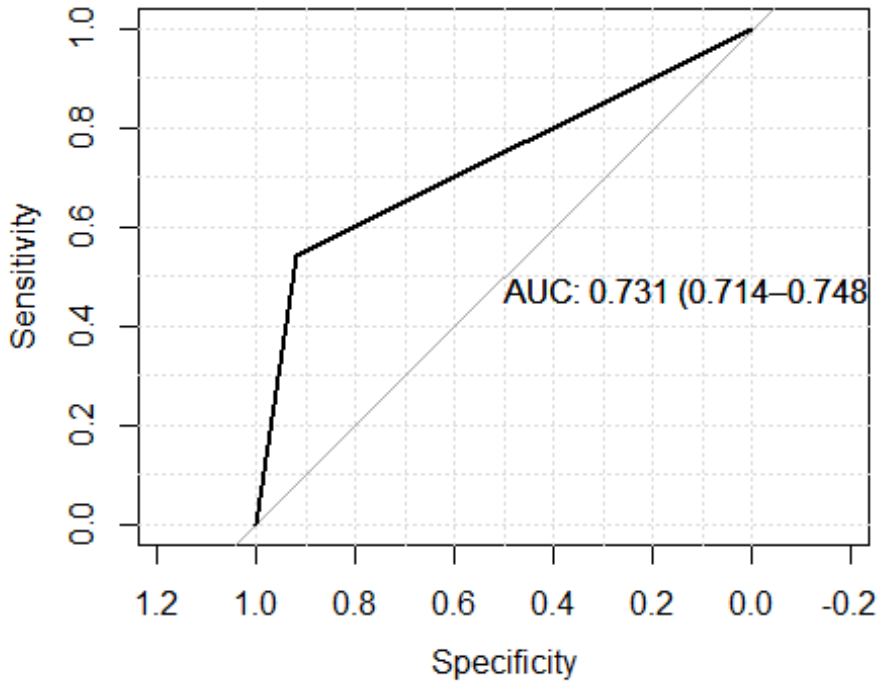
```
##
## ###################################
##
## Matrice de confusion :
##
##           Reference
## Prediction    0    1
##          0 3135  449
##          1  145  458
##
## ###################################
##
## Accuracy :  85.81 %
## Precision :  75.95 %
## Recall :  50.5 %
## F1 score :  60.66 %
##
## ###################################################################
##
## ALGORITHME N° : 7
##
## ###################################
##
## svm(formula = RainTomorrow ~ ., data = train_final, kernel = "polynomial",
##     type = "C-classification", cost = i, gamma = j, degree = k,
##     coef0 = l, scale = FALSE)
##
## ###################################
##
## Hyper-paramètres :
## Cost = 1
## Gamma = 1
## Nombre de degrés = 2
## Coef0 = 0.1
##
## ###################################
##
## Matrice de confusion :
##
##           Reference
## Prediction    0    1
##          0 3024  418
##          1  256  489
##
## ###################################
##
## Accuracy :  83.9 %
## Precision :  65.64 %
## Recall :  53.91 %
## F1 score :  59.2 %
```

```
##
## ###################################################################
##
## ALGORITHME N° : 8
##
## ##################################
##
## svm(formula = RainTomorrow ~ ., data = train_final, kernel = "polynomial",
##     type = "C-classification", cost = i, gamma = j, degree = k,
##     coef0 = l, scale = FALSE)
##
## ##################################
##
## Hyper-paramètres :
## Cost = 1
## Gamma = 1
## Nombre de degrés = 2
## Coef0 = 1
##
## ##################################
##
## Matrice de confusion :
##
##           Reference
## Prediction    0    1
##          0 3018  416
##          1  262  491
##
## ##################################
##
## Accuracy :  83.81 %
## Precision :  65.21 %
## Recall :  54.13 %
## F1 score :  59.16 %
##
## ###################################################################
```

Courbe ROC pour SVM Polynomial :

```
svm_fit_poly_final <- svm(formula = RainTomorrow~., data = train_final,
kernel = "polynomial", type = "C-classification", scale = FALSE, cost = 1,
gamma = 1, degree = 2, coef0 = 1)
pred_poly <- predict(svm_fit_poly_final, test_final, type = "class")

# Courbe ROC
pred_poly <- as.integer(pred_poly)
ROC_svm_poly <- roc(y_test_final,pred_poly,
            ci=TRUE, ci.alpha=0.9, stratified=FALSE,
            plot=TRUE, grid=TRUE,
            print.auc=TRUE)
```

```
## Warning in roc.default(y_test_final, pred_poly, ci = TRUE, ci.alpha = 0.9,
:
## Deprecated use a matrix as response. Unexpected results may be produced,
please
## pass a vector or factor.
```




```
print(ROC_svm_poly)

##
## Call:
## roc.default(response = y_test_final, predictor = pred_poly, ci = TRUE,
plot = TRUE, ci.alpha = 0.9, stratified = FALSE, grid = TRUE,     print.auc =
TRUE)
##
## Data: pred_poly in 3280 controls (y_test_final 0) < 907 cases
(y_test_final 1).
## Area under the curve: 0.7307
## 95% CI: 0.7139-0.7476 (DeLong)
```

Les courbes AUC présentent les mêmes résultats. De ce fait, en comparant les rappels et taux de reconnaissance. Le meilleur modèle est le modèle polynomial de degré 2.

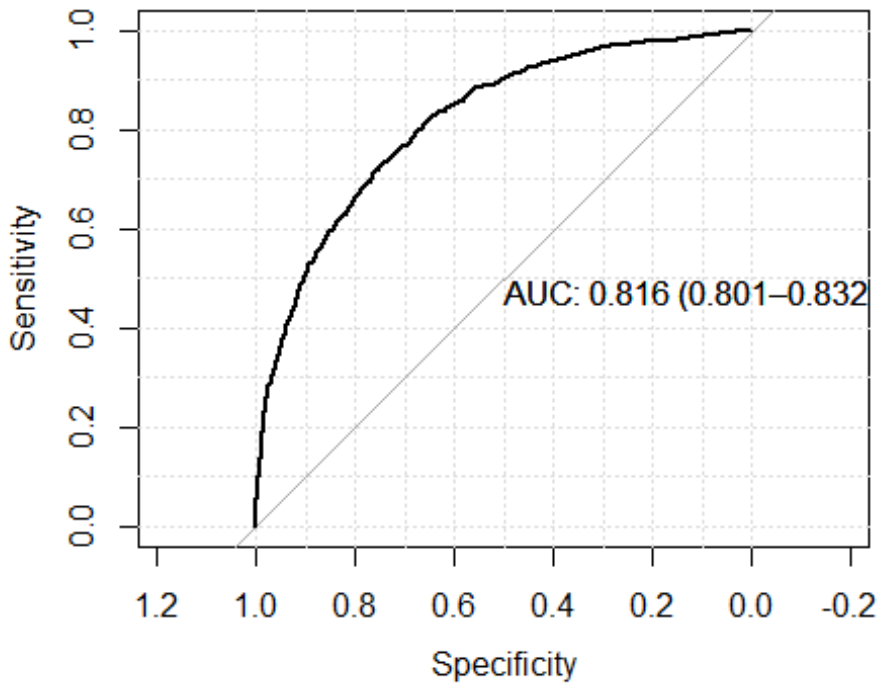
II) Réseau de neurones

```r
#########################################################################
## Mise en place d'un Réseaux de neurones avec 1 couche cachée

# Mis en forme du dataset pour l'algorithme des réseaux de neurones
dftrain = cbind.data.frame(X_train, y = y_train_final)

# Formula
var_exp = paste(colnames(X_train) , collapse = "+")
clas_cib = "y"
mod = paste(clas_cib, "~", var_exp, sep ="")
mod = as.formula(mod)

# Nombre de neurones dans la couche cachée
p = length(colnames(X_train)) - 1

#Réseau de neurones
nn <- neuralnet(mod, data=dftrain, hidden=p, linear.output=FALSE)

# Prédiction
nn.results <- compute(nn, X_test)

# Construction des résultats
results <- data.frame(actual = y_test_final, prediction =
nn.results$net.result)
roundedresults<-sapply(results,round,digits=0)
roundedresultsdf=data.frame(roundedresults)
attach(roundedresultsdf)

# Matrice de confusion
mc_nn <- confusionMatrix(factor(prediction), factor(actual), positive = "1")

# Affichage des données :
cat("#########################################################################\n")

## #####################################################################

cat("\n")

cat("Réseau de neurones")

## Réseau de neurones

cat("\n")

cat("###################################\n")

## ###################################

cat("\n")

print(nn$call)
```

```
## neuralnet(formula = mod, data = dftrain, hidden = p, linear.output =
FALSE)

cat("\n")

cat("Matrice de confusion : \n")

## Matrice de confusion :

cat("\n")

print(mc_nn$table)

##           Reference
## Prediction    0    1
##          0 2907  422
##          1  373  485

cat("\n")

cat("###################################\n")

## ###################################

cat("\n")

cat("Accuracy : ",round(mc_nn$overall[1]*100,2) ,"%\n")

## Accuracy :  81.01 %

cat("Precision : ",round(mc_nn$byClass[5]*100,2) ,"%\n")

## Precision :  56.53 %

cat("Recall : ",round(mc_nn$byClass[6]*100,2) ,"%\n")

## Recall :  53.47 %

cat("F1 score : ",round(mc_nn$byClass[7]*100,2) ,"%\n")

## F1 score :  54.96 %

cat("\n")

cat("#################################################################\n")

## #################################################################
```

Courbe ROC du réseaux de neuronnes avec la fonction d'activation par défaut :

```
# Courbe ROC
ROC_rnn <- roc(results$actual,results$prediction,
            ci=TRUE, ci.alpha=0.9, stratified=FALSE,
            plot=TRUE, grid=TRUE,
            print.auc=TRUE)
```




```
print(ROC_rnn)

##
## Call:
## roc.default(response = results$actual, predictor = results$prediction,
ci = TRUE, plot = TRUE, ci.alpha = 0.9, stratified = FALSE,     grid = TRUE,
print.auc = TRUE)
##
## Data: results$prediction in 3280 controls (results$actual 0) < 907 cases
(results$actual 1).
## Area under the curve: 0.8163
## 95% CI: 0.8008-0.8318 (DeLong)
```

Réseau de neurones avec fonction d'activation sigmoid :

```
nn_logistic <- neuralnet(mod, data=dftrain, hidden=p, linear.output=FALSE,
act.fct = 'logistic')

# Prédiction
nn_logistic.results <- compute(nn_logistic, X_test)

# Construction des résultats
results <- data.frame(actual = y_test_final, prediction =
nn_logistic.results$net.result)
roundedresults<-sapply(results,round,digits=0)
roundedresultsdf=data.frame(roundedresults)
attach(roundedresultsdf)
```

```r
# Matrice de confusion
mc_nn <- confusionMatrix(factor(prediction), factor(actual), positive = "1")

# Affichage des données :
cat("######################################################################\n")

## ######################################################################

cat("\n")

cat("Réseau de neurones")

## Réseau de neurones

cat("\n")

cat("###################################\n")

## ###################################

cat("\n")

print(nn$call)

## neuralnet(formula = mod, data = dftrain, hidden = p, linear.output =
## FALSE)

cat("\n")

cat("Matrice de confusion : \n")

## Matrice de confusion :

cat("\n")

print(mc_nn$table)

##           Reference
## Prediction    0    1
##          0 2928  425
##          1  352  482

cat("\n")

cat("###################################\n")

## ###################################

cat("\n")

cat("Accuracy : ",round(mc_nn$overall[1]*100,2) ,"%\n")

## Accuracy :  81.44 %
```

```
cat("Precision : ",round(mc_nn$byClass[5]*100,2) ,"%\n")
```

```
## Precision :  57.79 %
```

```
cat("Recall : ",round(mc_nn$byClass[6]*100,2) ,"%\n")
```

```
## Recall :  53.14 %
```

```
cat("F1 score : ",round(mc_nn$byClass[7]*100,2) ,"%\n")
```

```
## F1 score :  55.37 %
```

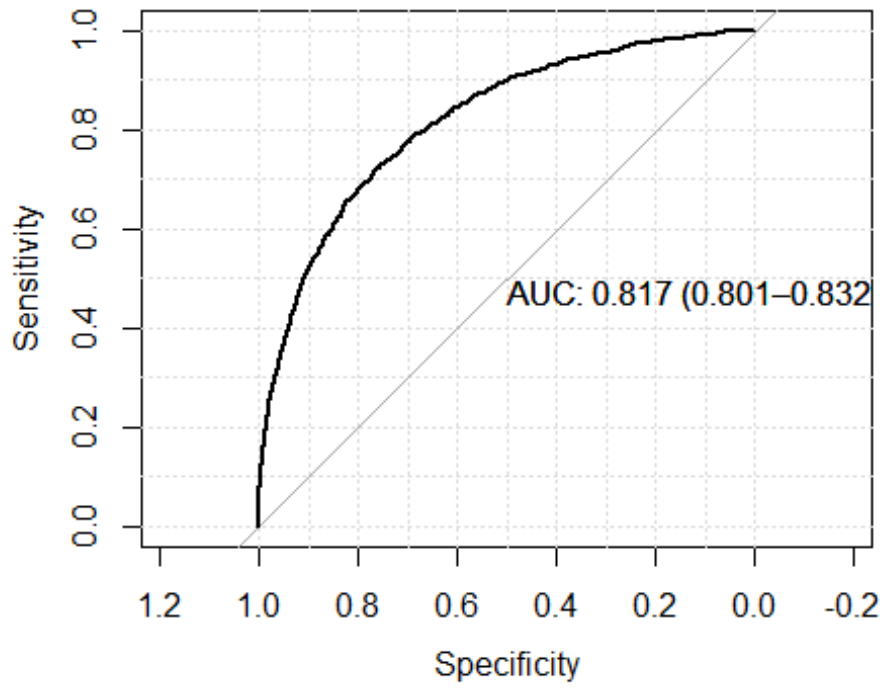
```
cat("\n")
```

```
cat("######################################################################\n")
```

```
## ######################################################################
```

Courbe ROC du réseaux de neuronnes avec la fonction d'activation sigmoid :

```
# Courbe ROC
ROC_rnn_logistic <- roc(results$actual,results$prediction,
            ci=TRUE, ci.alpha=0.9, stratified=FALSE,
            plot=TRUE, grid=TRUE,
            print.auc=TRUE)
```




```
print(ROC_rnn)
```

```
##
## Call:
## roc.default(response = results$actual, predictor = results$prediction,
ci = TRUE, plot = TRUE, ci.alpha = 0.9, stratified = FALSE,     grid = TRUE,
print.auc = TRUE)
##
## Data: results$prediction in 3280 controls (results$actual 0) < 907 cases
(results$actual 1).
## Area under the curve: 0.8163
## 95% CI: 0.8008-0.8318 (DeLong)
```

III) REGRESSION LOGISTIQUE PENALISEE

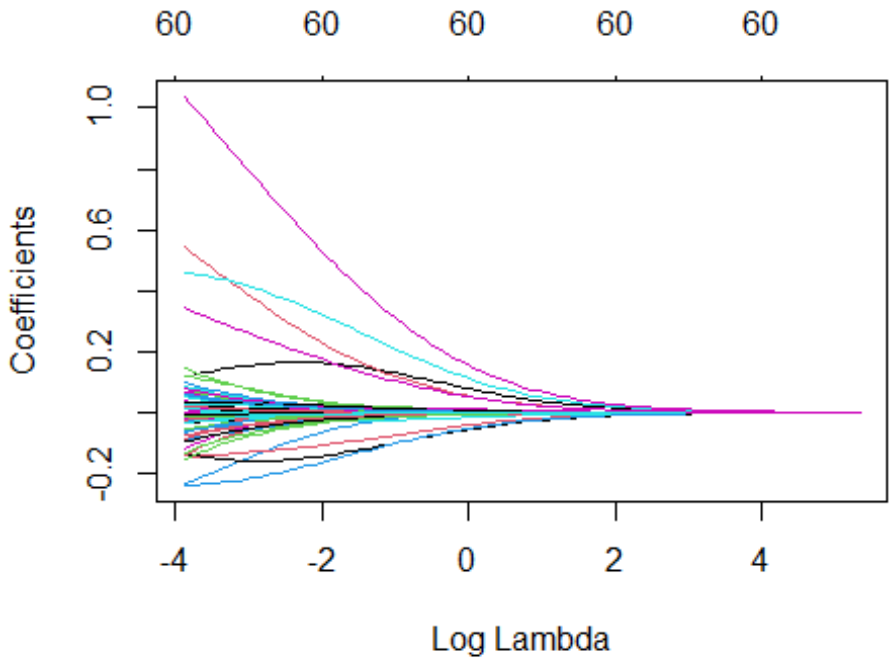
Nous allons maintenant tester les performances de la régression logistique binaire pénalisée. Nous allons mesurer et comparer les effets des pénalisations RIGDE, LASSO et ELASTICNET.

a) Pénalité Ridge

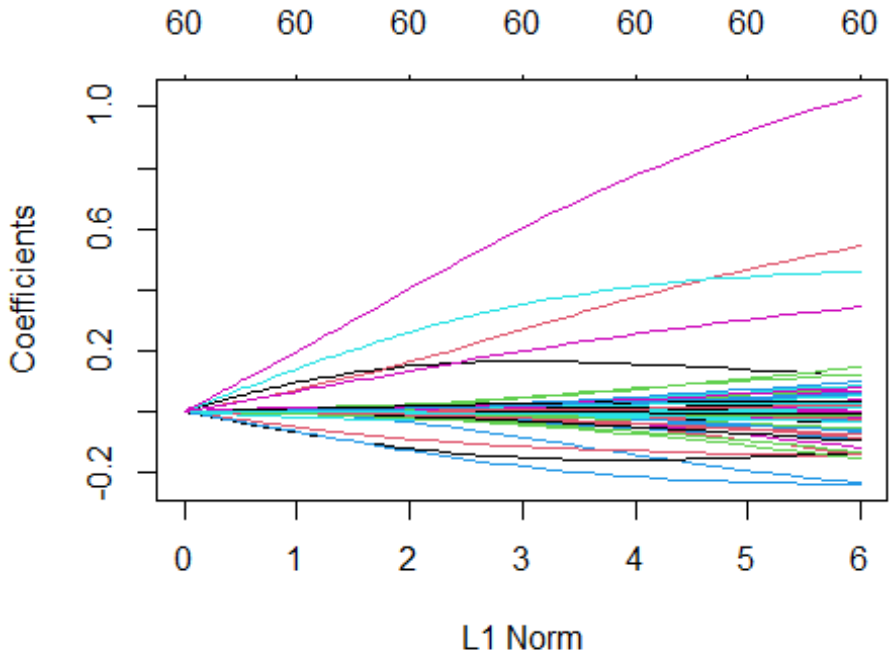
```
# Recodage des y_train
y_train_rl <- ifelse(y_train == "Yes", 1, 0)
y_test_rl <- ifelse(y_test == "Yes", 1, 0)
```

On entraine le modèle avec alpha = 0.Nous obtenons en sortie le chemin de régularisation. Nous voyons bien que les coefficients ne sont jamais nuls. La pénalité Ridge ne fait pas de sélection de variables, le nombre de coefficient ne change pas quelque soit la valeur de lambda.

```
# Régression logistique pénalisée ridge
modele_ridge <- glmnet(x = X_train, y = y_train_rl, family = "binomial",
alpha = 0, standardize = FALSE)

#Chemin de régularisation en fonction de lambda
plot(modele_ridge, xvar="lambda")
```




```r
#Chemin de régularisation pour la norme L1
plot(modele_ridge)
```



```r
# Nombre de variables sélectionnées vs. lambda avec alpha = 0
print(cbind(modele_ridge$lambda, modele_ridge$df))
```

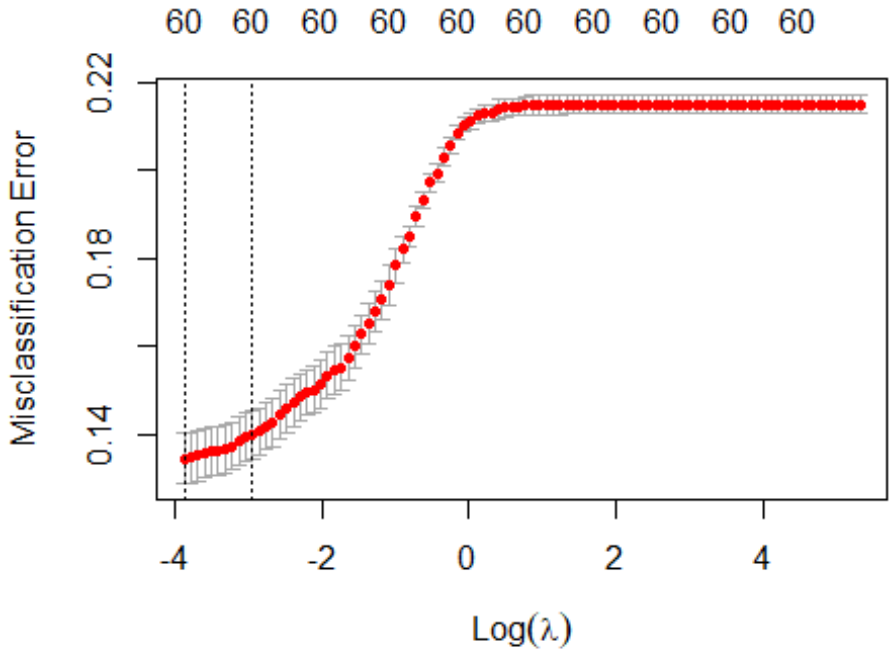
```
##                [,1] [,2]
##   [1,] 207.97433796   60
##   [2,] 189.49847099   60
##   [3,] 172.66394912   60
##   [4,] 157.32495978   60
##   [5,] 143.34864397   60
##   [6,] 130.61394554   60
##   [7,] 119.01056261   60
##   [8,] 108.43799225   60
##   [9,]  98.80465989   60
##  [10,]  90.02712623   60
##  [11,]  82.02936447   60
##  [12,]  74.74210182   60
##  [13,]  68.10221950   60
##  [14,]  62.05220602   60
##  [15,]  56.53965906   60
##  [16,]  51.51683159   60
##  [17,]  46.94021826   60
##  [18,]  42.77017865   60
##  [19,]  38.97059386   60
##  [20,]  35.50855371   60
##  [21,]  32.35407167   60
##  [22,]  29.47982511   60
##  [23,]  26.86091870   60
##  [24,]  24.47466872   60
##  [25,]  22.30040661   60
##  [26,]  20.31929995   60
##  [27,]  18.51418935   60
##  [28,]  16.86943979   60
##  [29,]  15.37080526   60
##  [30,]  14.00530528   60
##  [31,]  12.76111256   60
##  [32,]  11.62745049   60
##  [33,]  10.59449984   60
##  [34,]   9.65331367   60
##  [35,]   8.79573989   60
##  [36,]   8.01435061   60
##  [37,]   7.30237779   60
##  [38,]   6.65365467   60
##  [39,]   6.06256233   60
##  [40,]   5.52398100   60
##  [41,]   5.03324575   60
##  [42,]   4.58610607   60
##  [43,]   4.17868905   60
##  [44,]   3.80746583   60
##  [45,]   3.46922106   60
##  [46,]   3.16102502   60
```

```
##  [47,]   2.88020827   60
##  [48,]   2.62433851   60
##  [49,]   2.39119951   60
##  [50,]   2.17877194   60
##  [51,]   1.98521584   60
##  [52,]   1.80885474   60
##  [53,]   1.64816107   60
##  [54,]   1.50174298   60
##  [55,]   1.36833227   60
##  [56,]   1.24677341   60
##  [57,]   1.13601349   60
##  [58,]   1.03509319   60
##  [59,]   0.94313836   60
##  [60,]   0.85935255   60
##  [61,]   0.78301004   60
##  [62,]   0.71344958   60
##  [63,]   0.65006869   60
##  [64,]   0.59231838   60
##  [65,]   0.53969845   60
##  [66,]   0.49175312   60
##  [67,]   0.44806713   60
##  [68,]   0.40826208   60
##  [69,]   0.37199320   60
##  [70,]   0.33894635   60
##  [71,]   0.30883529   60
##  [72,]   0.28139922   60
##  [73,]   0.25640049   60
##  [74,]   0.23362257   60
##  [75,]   0.21286819   60
##  [76,]   0.19395756   60
##  [77,]   0.17672691   60
##  [78,]   0.16102698   60
##  [79,]   0.14672178   60
##  [80,]   0.13368743   60
##  [81,]   0.12181100   60
##  [82,]   0.11098965   60
##  [83,]   0.10112964   60
##  [84,]   0.09214556   60
##  [85,]   0.08395960   60
##  [86,]   0.07650086   60
##  [87,]   0.06970473   60
##  [88,]   0.06351236   60
##  [89,]   0.05787010   60
##  [90,]   0.05272908   60
##  [91,]   0.04804477   60
##  [92,]   0.04377660   60
##  [93,]   0.03988761   60
##  [94,]   0.03634411   60
##  [95,]   0.03311540   60
##  [96,]   0.03017352   60
```

```
##  [97,]   0.02749298   60
##  [98,]   0.02505058   60
##  [99,]   0.02282516   60
## [100,]   0.02079743   60
```

Pour trouver la valeur idéale de lambda, nous utilisons la validation croisée. Lambda.min est la valeur de lambda pour laquelle l'erreur est minimisée. Lambda.1se est la valeur pour laquelle le modèle est le plus régularisé. Cette valeur se situe à un écart-type de lambda.min.

```
#Entraînement du modèle
set.seed(1)
lambda.ridge <- cv.glmnet(X_train, y_train, family="binomial",
type.measure="class", nfolds=5, alpha=0, keep=TRUE)

#Evolution du lambda, le lambda idéal est celui minimise l'erreur. Il
apparaît au niveau des pointillés
plot(lambda.ridge)
```




```
#valeur de lambda.min et lambda.1se
cat("lambda.min :", lambda.ridge$lambda.min,"\n","lambda.1se :",
lambda.ridge$lambda.1se)
```

```
## lambda.min : 0.02065428
##  lambda.1se : 0.05236614
```

Nous réalisons maintenant les prédictions avec les deux valeurs de lambda.

```r
#prédiction avec lambda.min qui minimise l'erreur
pred_ridge_min <- predict(modele_ridge, X_test, s=c(lambda.ridge$lambda.min),
type="class")
mc_ridge_min <-  confusionMatrix(factor(pred_ridge_min), factor(y_test_rl),
positive = "1")

cat("###################################\n")

## ###################################

cat("Matrice de confusion pour lambda.1se \n")

## Matrice de confusion pour lambda.1se

cat("###################################\n")

## ###################################

print(mc_ridge_min)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3161  515
##          1  119  392
##
##                Accuracy : 0.8486
##                  95% CI : (0.8374, 0.8593)
##     No Information Rate : 0.7834
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4702
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.43219
##             Specificity : 0.96372
##          Pos Pred Value : 0.76712
##          Neg Pred Value : 0.85990
##              Prevalence : 0.21662
##          Detection Rate : 0.09362
##    Detection Prevalence : 0.12204
##       Balanced Accuracy : 0.69796
##
##        'Positive' Class : 1
##

cat("\n")

#Prédiction avec lambda.1se qui est le modèle le plus régularisé
pred_ridge_1se <- predict(modele_ridge, X_test, s=c(lambda.ridge$lambda.1se),
```

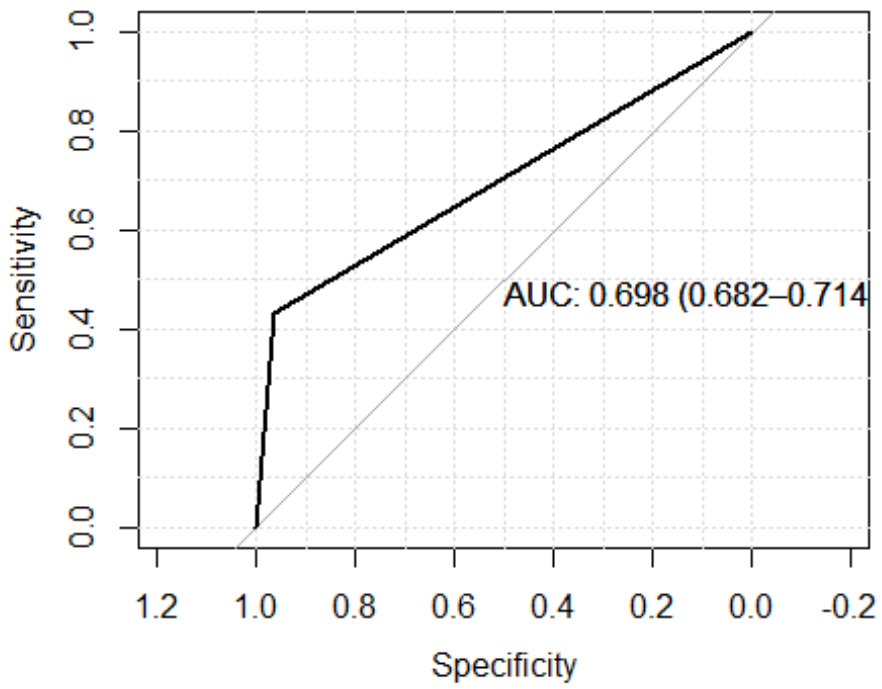
```
type="class")
mc_ridge_1se <-  confusionMatrix(factor(pred_ridge_1se), factor(y_test_rl),
positive = "1")

cat("###################################\n")

## ###################################

cat("Matrice de confusion pour lambda.1se \n")

## Matrice de confusion pour lambda.1se

cat("###################################\n")

## ###################################

print(mc_ridge_1se)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3190  574
##          1   90  333
##
##                Accuracy : 0.8414
##                  95% CI : (0.83, 0.8524)
##     No Information Rate : 0.7834
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.421
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.36714
##             Specificity : 0.97256
##          Pos Pred Value : 0.78723
##          Neg Pred Value : 0.84750
##              Prevalence : 0.21662
##          Detection Rate : 0.07953
##    Detection Prevalence : 0.10103
##       Balanced Accuracy : 0.66985
##
##        'Positive' Class : 1
##
```

En comparant les performances, on se rend compte que les différences sont minimes. Cela se joue sur la précision et le rappel. Avec lambda.min, les prédictions erronnées d'absence de pluie sont légèrement inférieures (300) à celles obtenues avec lambda.1se (318). Lambda.min nous semble meilleure même si la précision est légèrement inférieure.

Courbe ROC

```r
# Courbe ROC
pred_ridge_min <- as.integer(pred_ridge_min)
ROC_ridge <- roc(y_test_rl,pred_ridge_min,
            ci=TRUE, ci.alpha=0.9, stratified=FALSE,
            plot=TRUE, grid=TRUE,
            print.auc=TRUE)
```

```
## Warning in roc.default(y_test_rl, pred_ridge_min, ci = TRUE, ci.alpha =
0.9, :
## Deprecated use a matrix as response. Unexpected results may be produced,
please
## pass a vector or factor.
```



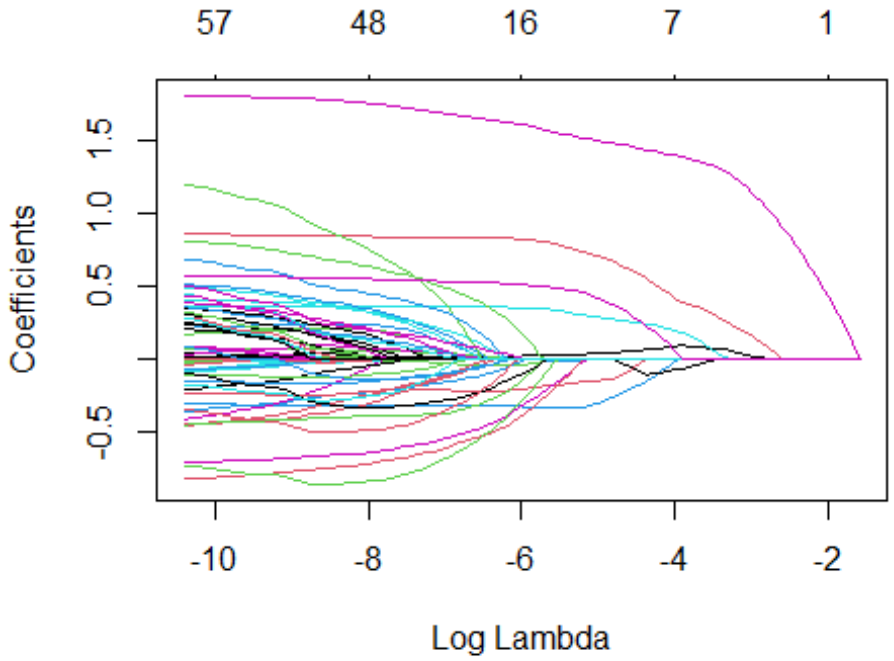

```r
print(ROC_ridge)
```

```
##
## Call:
## roc.default(response = y_test_rl, predictor = pred_ridge_min,     ci =
TRUE, plot = TRUE, ci.alpha = 0.9, stratified = FALSE,     grid = TRUE,
print.auc = TRUE)
##
## Data: pred_ridge_min in 3280 controls (y_test_rl 0) < 907 cases (y_test_rl
1).
## Area under the curve: 0.698
## 95% CI: 0.6815-0.7144 (DeLong)
```

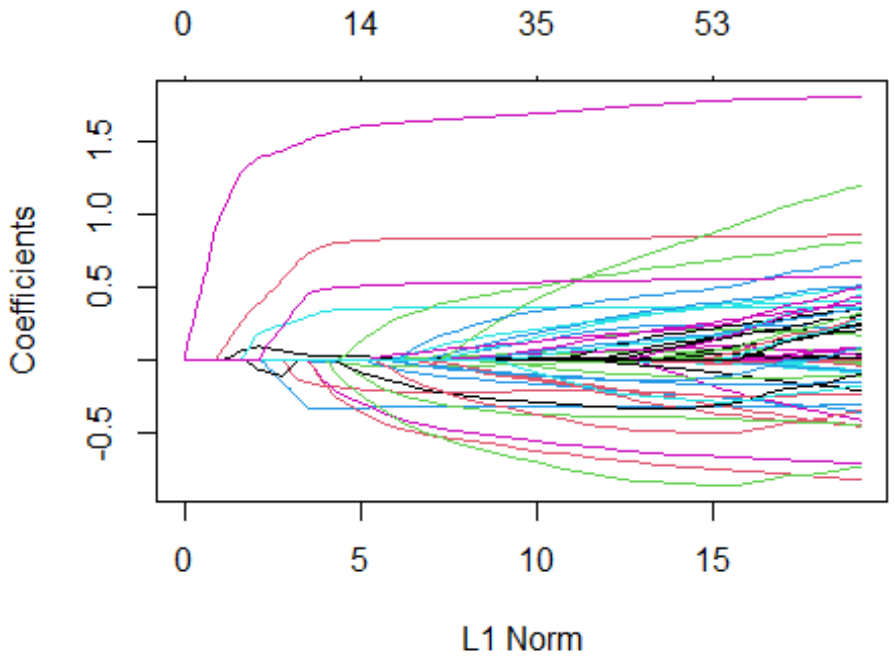
b) Régression Lasso

On entraine le modèle avec alpha = 1. Nous obtenons en sortie le chemin de régularisation. Ici, il apparaît clairement que certains coefficients sont nullifiés par la pénalité Lasso. On peut voir que selon la valeur de lambda, jusqu'à 99 variables peuvent être supprimées.

```
# Régression logistique pénalisée ridge
modele_lasso <- glmnet(x = X_train, y = y_train_rl, family = "binomial",
alpha = 1, standardize = FALSE)

#Chemin de régularisation en fonction de lambda
plot(modele_lasso, xvar="lambda")
```




```
#Chemin de régularisation pour la norme L2
plot(modele_lasso)
```




```
# Nombre de variables sélectionnées vs. lambda avec alpha = 1
print(cbind(modele_lasso$lambda, modele_lasso$df))
```

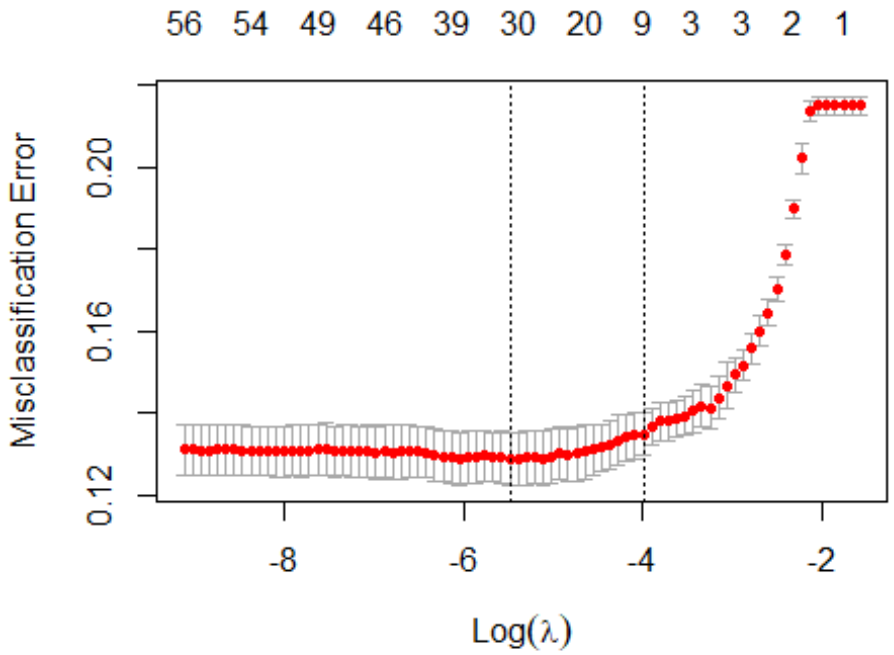
```
##                [,1] [,2]
##  [1,] 2.079743e-01    0
##  [2,] 1.894985e-01    1
##  [3,] 1.726639e-01    1
##  [4,] 1.573250e-01    1
##  [5,] 1.433486e-01    1
##  [6,] 1.306139e-01    1
##  [7,] 1.190106e-01    1
##  [8,] 1.084380e-01    1
##  [9,] 9.880466e-02    1
## [10,] 9.002713e-02    1
## [11,] 8.202936e-02    1
## [12,] 7.474210e-02    1
## [13,] 6.810222e-02    2
## [14,] 6.205221e-02    3
## [15,] 5.653966e-02    3
## [16,] 5.151683e-02    3
## [17,] 4.694022e-02    3
## [18,] 4.277018e-02    3
## [19,] 3.897059e-02    3
## [20,] 3.550855e-02    4
## [21,] 3.235407e-02    4
## [22,] 2.947983e-02    5
## [23,] 2.686092e-02    5
```

```
## [24,] 2.447467e-02    5
## [25,] 2.230041e-02    5
## [26,] 2.031930e-02    6
## [27,] 1.851419e-02    7
## [28,] 1.686944e-02    7
## [29,] 1.537081e-02    7
## [30,] 1.400531e-02    7
## [31,] 1.276111e-02    8
## [32,] 1.162745e-02    8
## [33,] 1.059450e-02    8
## [34,] 9.653314e-03    8
## [35,] 8.795740e-03    8
## [36,] 8.014351e-03    7
## [37,] 7.302378e-03    7
## [38,] 6.653655e-03    7
## [39,] 6.062562e-03    7
## [40,] 5.523981e-03    9
## [41,] 5.033246e-03   10
## [42,] 4.586106e-03   10
## [43,] 4.178689e-03   10
## [44,] 3.807466e-03   11
## [45,] 3.469221e-03   12
## [46,] 3.161025e-03   14
## [47,] 2.880208e-03   14
## [48,] 2.624339e-03   14
## [49,] 2.391200e-03   16
## [50,] 2.178772e-03   18
## [51,] 1.985216e-03   21
## [52,] 1.808855e-03   22
## [53,] 1.648161e-03   23
## [54,] 1.501743e-03   28
## [55,] 1.368332e-03   29
## [56,] 1.246773e-03   29
## [57,] 1.136013e-03   32
## [58,] 1.035093e-03   34
## [59,] 9.431384e-04   35
## [60,] 8.593525e-04   35
## [61,] 7.830100e-04   35
## [62,] 7.134496e-04   36
## [63,] 6.500687e-04   36
## [64,] 5.923184e-04   36
## [65,] 5.396984e-04   38
## [66,] 4.917531e-04   39
## [67,] 4.480671e-04   40
## [68,] 4.082621e-04   43
## [69,] 3.719932e-04   48
## [70,] 3.389464e-04   48
## [71,] 3.088353e-04   48
## [72,] 2.813992e-04   50
## [73,] 2.564005e-04   49
```

```
## [74,] 2.336226e-04   51
## [75,] 2.128682e-04   52
## [76,] 1.939576e-04   53
## [77,] 1.767269e-04   54
## [78,] 1.610270e-04   55
## [79,] 1.467218e-04   56
## [80,] 1.336874e-04   57
## [81,] 1.218110e-04   57
## [82,] 1.109896e-04   58
## [83,] 1.011296e-04   57
## [84,] 9.214556e-05   56
## [85,] 8.395960e-05   57
## [86,] 7.650086e-05   57
## [87,] 6.970473e-05   57
## [88,] 6.351236e-05   58
## [89,] 5.787010e-05   57
## [90,] 5.272908e-05   57
## [91,] 4.804477e-05   57
## [92,] 4.377660e-05   57
## [93,] 3.988761e-05   57
## [94,] 3.634411e-05   57
## [95,] 3.311540e-05   56
## [96,] 3.017352e-05   57
```

On utilise cv.glmnet pour estimer lambda en validation croisée. Les valeurs sont faibles mais nous remarquons un facteur 4.

```
#Entraînement du modèle
set.seed(1)
lambda.lasso <- cv.glmnet(X_train, y_train_rl, family="binomial",
type.measure="class", nfolds=5, alpha=1, keep=TRUE)

#Evolution du lambda, le lambda idéal est celui minimise l'erreur. Il
apparaît au niveau des pointillés
plot(lambda.lasso)
```




```
#valeur de lambda.min et lambda.1se
cat("lambda.min :", lambda.lasso$lambda.min, "\n", "lambda.1se :",
lambda.lasso$lambda.1se)

## lambda.min : 0.004149927
##  lambda.1se : 0.01838675
```

Prédiction avec les valeurs de lambda.

```
#prédiction avec lambda.min qui minimise l'erreur
pred_lasso_min <- predict(modele_lasso, X_test, s=c(lambda.lasso$lambda.min),
type="class")
mc_lasso_min <-  confusionMatrix(factor(pred_lasso_min), factor(y_test_rl),
positive = "1")

cat("###################################\n")

## ###################################

cat("Matrice de confusion pour lambda.min \n")

## Matrice de confusion pour lambda.min

cat("###################################\n")

## ###################################

print(mc_lasso_min)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3127  478
##          1  153  429
##
##                Accuracy : 0.8493
##                  95% CI : (0.8381, 0.86)
##     No Information Rate : 0.7834
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4898
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.4730
##             Specificity : 0.9534
##          Pos Pred Value : 0.7371
##          Neg Pred Value : 0.8674
##              Prevalence : 0.2166
##          Detection Rate : 0.1025
##    Detection Prevalence : 0.1390
##       Balanced Accuracy : 0.7132
##
##        'Positive' Class : 1
##

cat("\n")

#Prédiction avec lambda.1se qui est le modèle le plus régularisé
pred_lasso_1se <- predict(modele_lasso, X_test, s=c(lambda.lasso$lambda.1se),
type="class")
mc_lasso_1se <-  confusionMatrix(data = factor(pred_lasso_1se), reference =
factor(y_test_rl), positive = "1")

cat("###################################\n")

## #################################

cat("Matrice de confusion pour lambda.1se \n")

## Matrice de confusion pour lambda.1se

cat("###################################\n")

## #################################

print(mc_lasso_1se)

## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction    0    1
##          0 3176  525
##          1  104  382
##
##                Accuracy : 0.8498
##                  95% CI : (0.8386, 0.8605)
##     No Information Rate : 0.7834
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4681
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.42117
##             Specificity : 0.96829
##          Pos Pred Value : 0.78601
##          Neg Pred Value : 0.85815
##              Prevalence : 0.21662
##          Detection Rate : 0.09123
##    Detection Prevalence : 0.11607
##       Balanced Accuracy : 0.69473
##
##        'Positive' Class : 1
##
```

On peut noter dans un premier temps que l'accuracy et les bonnes prédictions sont meilleures avec Lasso. La précision est également plus performante par rapport à Ridge.

En ce qui concerne les écarts entre lambda.min et lambda.1se, ils sont faibles mais c'est toujours avec lambda.min qu'on obtient les meilleurs scores globaux.

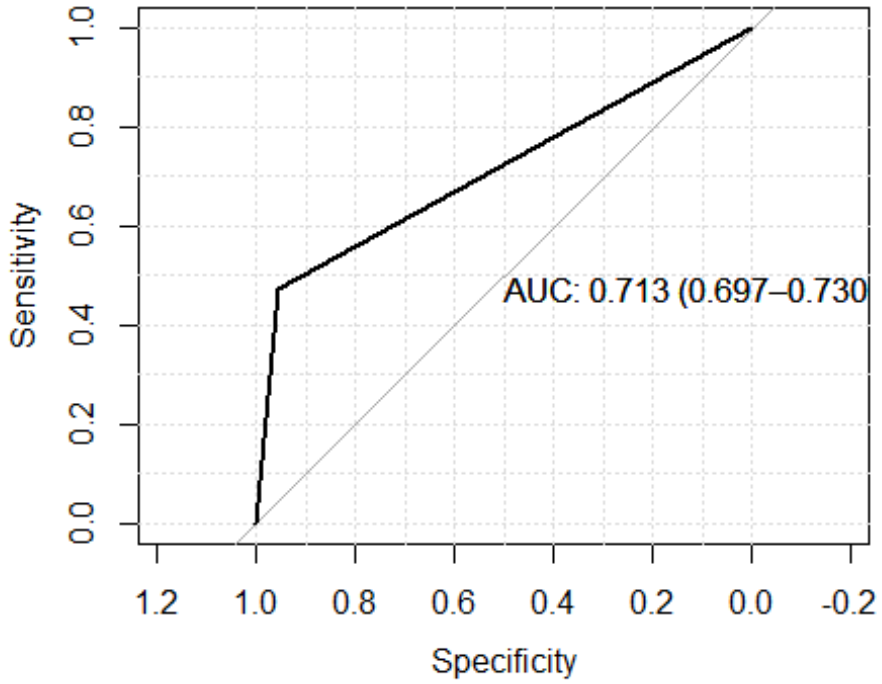
```
#Coeff du modèle
print(coef(lambda.lasso,s="lambda.min"))
```

```
## 61 x 1 sparse Matrix of class "dgCMatrix"
##                                    s1
## (Intercept)           -1.9682088994
## Rainfall               0.0125886672
## WindGustSpeed          0.7219238209
## WindSpeed9am           .
## WindSpeed3pm          -0.2199390579
## Humidity9am            0.2754842576
## Humidity3pm            1.6232740161
## Pressure9am            .
## Pressure3pm           -0.1454747549
## Temp9am                .
## Temp3pm                .
## Location_AliceSprings  .
## Location_Cairns       -0.4029438799
## Location_Perth         .
```

```
## Location_Sydney       -0.4440257909
## WindGustDir_E          .
## WindGustDir_ENE       -0.1674570999
## WindGustDir_ESE        .
## WindGustDir_N          .
## WindGustDir_NE         .
## WindGustDir_NNE       -0.1360115151
## WindGustDir_NNW        0.2121920868
## WindGustDir_NW         0.2516438494
## WindGustDir_S          .
## WindGustDir_SE         0.0140376764
## WindGustDir_SSE        .
## WindGustDir_SSW        .
## WindGustDir_SW        -0.2960061621
## WindGustDir_W          .
## WindGustDir_WSW        0.2109073100
## WindDir9am_E           .
## WindDir9am_ENE         .
## WindDir9am_ESE        -0.0534992540
## WindDir9am_N           0.5311453441
## WindDir9am_NE          0.1655122049
## WindDir9am_NNE         0.2022499149
## WindDir9am_NNW         .
## WindDir9am_NW          .
## WindDir9am_S           .
## WindDir9am_SE          .
## WindDir9am_SSE        -0.0307178352
## WindDir9am_SW          .
## WindDir9am_W           0.0246049196
## WindDir9am_WNW         .
## WindDir9am_WSW         .
## WindDir3pm_E           .
## WindDir3pm_ENE        -0.0738493640
## WindDir3pm_ESE         .
## WindDir3pm_N           .
## WindDir3pm_NE         -0.2453134246
## WindDir3pm_NNE         .
## WindDir3pm_NNW         0.7877709993
## WindDir3pm_NW          0.5370354933
## WindDir3pm_S           .
## WindDir3pm_SE          0.0588291996
## WindDir3pm_SSE         .
## WindDir3pm_SSW        -0.3029539708
## WindDir3pm_SW         -0.6073226365
## WindDir3pm_WNW         0.1899011054
## WindDir3pm_WSW        -0.0008903331
## RainToday_Yes          0.6097716240

# Courbe ROC
pred_lasso_min <- as.integer(pred_lasso_min)
```

```
ROC_lasso <- roc(y_test_rl,pred_lasso_min,
            ci=TRUE, ci.alpha=0.9, stratified=FALSE,
            plot=TRUE, grid=TRUE,
            print.auc=TRUE)
```

```
## Warning in roc.default(y_test_rl, pred_lasso_min, ci = TRUE, ci.alpha =
0.9, :
## Deprecated use a matrix as response. Unexpected results may be produced,
please
## pass a vector or factor.
```




```
print(ROC_lasso)
```

```
##
## Call:
## roc.default(response = y_test_rl, predictor = pred_lasso_min,     ci =
TRUE, plot = TRUE, ci.alpha = 0.9, stratified = FALSE,     grid = TRUE,
print.auc = TRUE)
##
## Data: pred_lasso_min in 3280 controls (y_test_rl 0) < 907 cases (y_test_rl
1).
## Area under the curve: 0.7132
## 95% CI: 0.6965-0.7298 (DeLong)
```

On peut voir que le modèle en Lasso a été simplifié puisque 29 coefficients sont nuls : 3 appartenant à la variable “Location” et 3 autres pour la variable “WindGustDir”. Ces variables ont été recodées.

L'AUC est meilleure en Lasso par rapport à Ridge. 0.713 vs 0.698.

c) Pénalité ElasticNet

Nous allons désormais entraîner un modèle en combinant les deux pénalités Ridge et Lasso. C'est la pénalité ElasticNet, qui tire le meilleur parti des deux pénalités en effaçant leurs défauts respectifs. Les meilleures performances en Lasso nous donne une première intuition qui est que les performances devraient être meilleures pour 0.5 <= alpha < 1. Vérifions.

```
# Régression logistique pénalisée elasticnet

alpha <- c(seq(0.0, 1, 0.1))
precision <- c()
rappel <- c()


for (i in alpha){
  modele <- cv.glmnet(x = X_train, y = y_train_rl, family = "binomial", alpha
= i, nfolds = 5)

# Prédiction
  prediction <- predict(modele, X_test, type = "class", s=c(0))

  precision <- append(precision, precision(factor(prediction),
factor(y_test_rl), relevant = "1"))
  rappel <- append(rappel, recall(factor(prediction), factor(y_test_rl),
relevant = "1"))
  # Matrice de confusion
  # mc_elet <-  table(factor(prediction), factor(y_test_rl))
  # cat("La matrice de confusion pour alpha =", i, "est :\n", mc_elet, "\n")
  }

comparaison <- as.data.frame(cbind(alpha, precision, rappel))
print(comparaison)
```

```
##    alpha precision    rappel
## 1    0.0 0.7609489 0.4597574
## 2    0.1 0.7257053 0.5104741
## 3    0.2 0.7245696 0.5104741
## 4    0.3 0.7234375 0.5104741
## 5    0.4 0.7230047 0.5093716
## 6    0.5 0.7230047 0.5093716
## 7    0.6 0.7230047 0.5093716
## 8    0.7 0.7230047 0.5093716
## 9    0.8 0.7230047 0.5093716
## 10   0.9 0.7230047 0.5093716
## 11   1.0 0.7230047 0.5093716
```

On voit ici que la meilleure précision est avec la pénalité Ridge. Cependant ce qui importe dans ce contexte est le rappel puisqu'on souhaite éviter de prédire un jour sec au lieu d'un jour pluvieux. Sur ce critère, c'est bien lorsqu'on tend vers Lasso qu'on obtient les meilleurs résultats. A l'issue de la validation croisée, 2 valeurs de alpha semblent être les meilleurs compromis : alpha = 0.6 et alpha = 0.9. Va comparer les deux modèles.

```
# Régression logistique pénalisée elasticnet
modele_0.2 <- glmnet(x = X_train, y = y_train_rl, family = "binomial", alpha
= 0.2, standardize = FALSE)
# plot(modele, xvar="lambda")
# plot(modele)

# Prédiction
pred0.2 <- predict(modele_0.2, X_test, type = "class", s=c(0))

# Matrice de confusion
mc0.2 <-  confusionMatrix(factor(pred0.2), factor(y_test_rl), positive = "1")
print(mc0.2)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3106  445
##          1  174  462
##
##                Accuracy : 0.8522
##                  95% CI : (0.841, 0.8628)
##     No Information Rate : 0.7834
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5116
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.5094
##             Specificity : 0.9470
##          Pos Pred Value : 0.7264
##          Neg Pred Value : 0.8747
##              Prevalence : 0.2166
##          Detection Rate : 0.1103
##    Detection Prevalence : 0.1519
##       Balanced Accuracy : 0.7282
##
##        'Positive' Class : 1
##
```

```
# Régression logistique pénalisée elasticnet
modele_0.9 <- glmnet(x = X_train, y = y_train_rl, family = "binomial", alpha
= 0.9, standardize = FALSE)
```

```r
# plot(modele, xvar="lambda")
# plot(modele)

# Prédiction
pred0.9 <- predict(modele_0.9, X_test, type = "class", s=c(0))

# Matrice de confusion
mc0.9 <-  confusionMatrix(factor(pred0.9), factor(y_test_rl), positive = "1")
print(mc0.9)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3105  445
##          1  175  462
##
##                Accuracy : 0.8519
##                  95% CI : (0.8408, 0.8626)
##     No Information Rate : 0.7834
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.511
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.5094
##             Specificity : 0.9466
##          Pos Pred Value : 0.7253
##          Neg Pred Value : 0.8746
##              Prevalence : 0.2166
##          Detection Rate : 0.1103
##    Detection Prevalence : 0.1521
##       Balanced Accuracy : 0.7280
##
##        'Positive' Class : 1
##
```

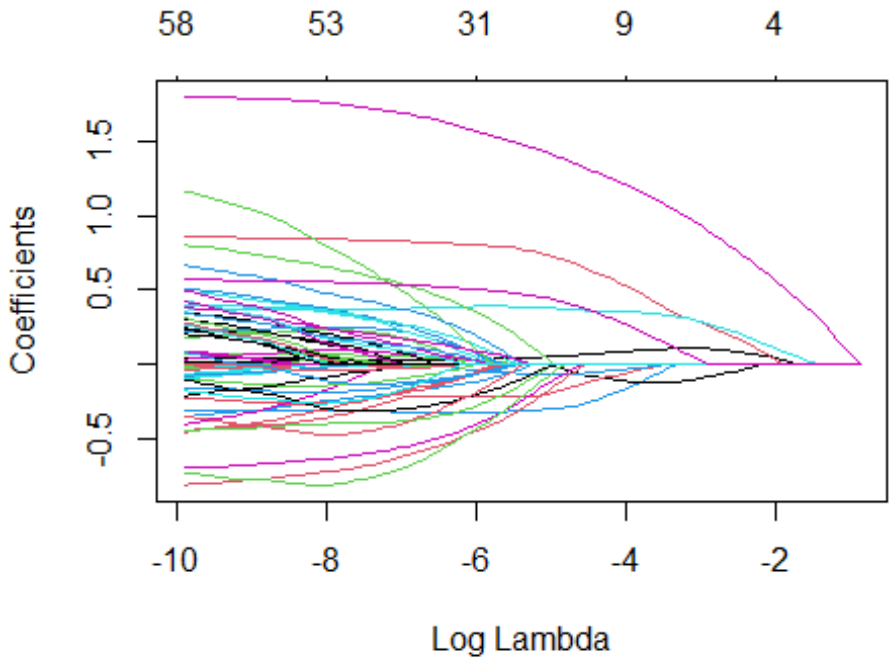
```r
# Régression logistique pénalisée elasticnet
modele_0.5 <- glmnet(x = X_train, y = y_train_rl, family = "binomial", alpha
= 0.5, standardize = FALSE)
# plot(modele, xvar="lambda")
# plot(modele)

# Prédiction
pred0.5 <- predict(modele_0.5, X_test, type = "class", s=c(0))

# Matrice de confusion
mc0.5 <-  confusionMatrix(factor(pred0.5), factor(y_test_rl), positive = "1")
print(mc0.5)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3105  444
##          1  175  463
##
##                Accuracy : 0.8522
##                  95% CI : (0.841, 0.8628)
##     No Information Rate : 0.7834
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5121
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.5105
##             Specificity : 0.9466
##          Pos Pred Value : 0.7257
##          Neg Pred Value : 0.8749
##              Prevalence : 0.2166
##          Detection Rate : 0.1106
##    Detection Prevalence : 0.1524
##       Balanced Accuracy : 0.7286
##
##        'Positive' Class : 1
##
```

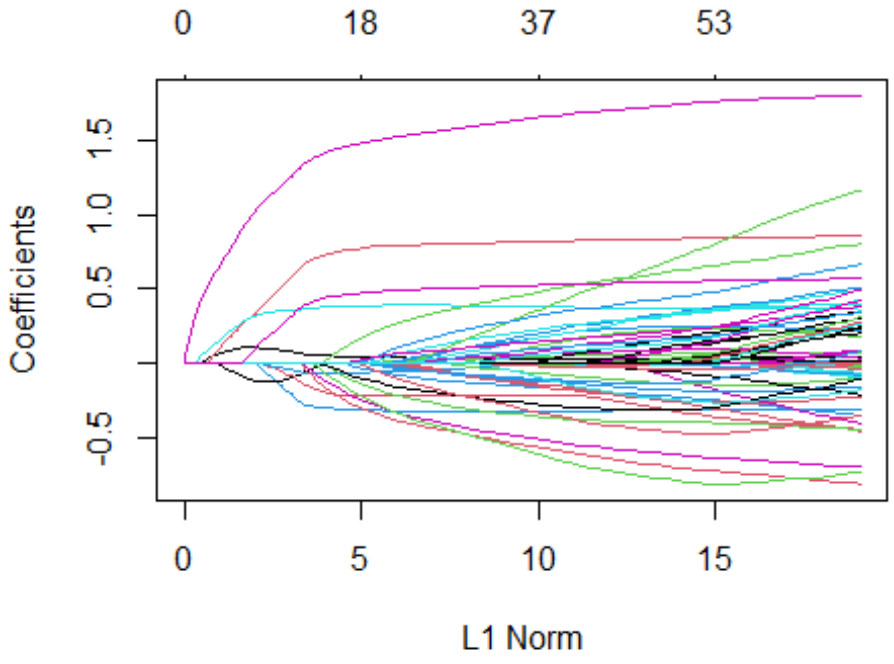
```
#Courbe ROC
# pred <- prediction(prediction, y_test_rl)
# perf <- performance(pred,"tpr","fpr")
# plot(perf,colorize=TRUE)

# pred_elet_min <- as.integer(pred_elet_min)
# ROC_elet <- roc(y_test_rl,pred_elet_min,
#             ci=TRUE, ci.alpha=0.9, stratified=FALSE,
#             plot=TRUE, grid=TRUE,
#             print.auc=TRUE)
# print(ROC_elet)
```

Les résultats sont équivalents à deux bonnes prédictions près. Par souci de performance, on choisit le paramètre alpha = 0.5. D'autant plus qu'il diminue davantage la complexité du modèle. Nous allons donc chercher la meilleure valeur de lambda.

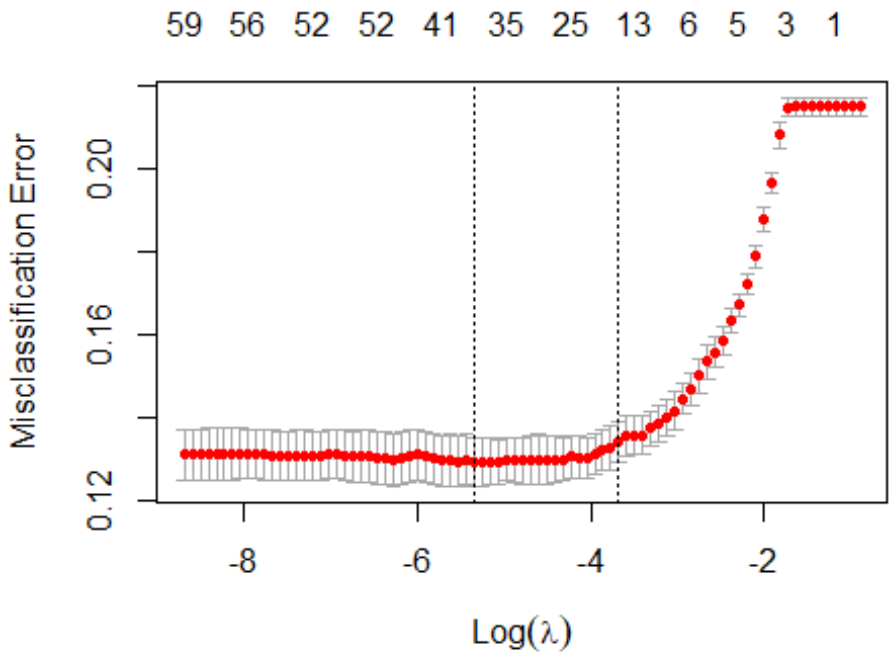
```
#Chemin de régularisation en fonction de lambda
plot(modele_0.5, xvar="lambda")
```




```r
#Chemin de régularisation pour ElasticNet
plot(modele_0.5)
```



```r
#Entraînement du modèle
set.seed(1)
lambda.elet <- cv.glmnet(X_train, y_train_rl, family="binomial",
type.measure="class", nfolds=5, alpha=0.5, keep=TRUE, foldid =
lambda.lasso$foldid)

#Evolution du lambda, le lambda idéal est celui minimise l'erreur. Il
apparaît au niveau des pointillés
plot(lambda.elet)
```




```r
#valeur de lambda.min et lambda.1se
cat("lambda.min :", lambda.elet$lambda.min, "\n", "lambda.1se :",
lambda.elet$lambda.1se)
```

```
## lambda.min : 0.004749481
##  lambda.1se : 0.02534655
```

```r
# Nombre de variables sélectionnées vs. lambda avec alpha = 0.5
print(cbind(modele_lasso$lambda, modele_lasso$df))
```

```
##               [,1] [,2]
##  [1,] 2.079743e-01    0
##  [2,] 1.894985e-01    1
##  [3,] 1.726639e-01    1
##  [4,] 1.573250e-01    1
##  [5,] 1.433486e-01    1
##  [6,] 1.306139e-01    1
```

```
##  [7,] 1.190106e-01    1
##  [8,] 1.084380e-01    1
##  [9,] 9.880466e-02    1
## [10,] 9.002713e-02    1
## [11,] 8.202936e-02    1
## [12,] 7.474210e-02    1
## [13,] 6.810222e-02    2
## [14,] 6.205221e-02    3
## [15,] 5.653966e-02    3
## [16,] 5.151683e-02    3
## [17,] 4.694022e-02    3
## [18,] 4.277018e-02    3
## [19,] 3.897059e-02    3
## [20,] 3.550855e-02    4
## [21,] 3.235407e-02    4
## [22,] 2.947983e-02    5
## [23,] 2.686092e-02    5
## [24,] 2.447467e-02    5
## [25,] 2.230041e-02    5
## [26,] 2.031930e-02    6
## [27,] 1.851419e-02    7
## [28,] 1.686944e-02    7
## [29,] 1.537081e-02    7
## [30,] 1.400531e-02    7
## [31,] 1.276111e-02    8
## [32,] 1.162745e-02    8
## [33,] 1.059450e-02    8
## [34,] 9.653314e-03    8
## [35,] 8.795740e-03    8
## [36,] 8.014351e-03    7
## [37,] 7.302378e-03    7
## [38,] 6.653655e-03    7
## [39,] 6.062562e-03    7
## [40,] 5.523981e-03    9
## [41,] 5.033246e-03   10
## [42,] 4.586106e-03   10
## [43,] 4.178689e-03   10
## [44,] 3.807466e-03   11
## [45,] 3.469221e-03   12
## [46,] 3.161025e-03   14
## [47,] 2.880208e-03   14
## [48,] 2.624339e-03   14
## [49,] 2.391200e-03   16
## [50,] 2.178772e-03   18
## [51,] 1.985216e-03   21
## [52,] 1.808855e-03   22
## [53,] 1.648161e-03   23
## [54,] 1.501743e-03   28
## [55,] 1.368332e-03   29
## [56,] 1.246773e-03   29
```

```
## [57,] 1.136013e-03   32
## [58,] 1.035093e-03   34
## [59,] 9.431384e-04   35
## [60,] 8.593525e-04   35
## [61,] 7.830100e-04   35
## [62,] 7.134496e-04   36
## [63,] 6.500687e-04   36
## [64,] 5.923184e-04   36
## [65,] 5.396984e-04   38
## [66,] 4.917531e-04   39
## [67,] 4.480671e-04   40
## [68,] 4.082621e-04   43
## [69,] 3.719932e-04   48
## [70,] 3.389464e-04   48
## [71,] 3.088353e-04   48
## [72,] 2.813992e-04   50
## [73,] 2.564005e-04   49
## [74,] 2.336226e-04   51
## [75,] 2.128682e-04   52
## [76,] 1.939576e-04   53
## [77,] 1.767269e-04   54
## [78,] 1.610270e-04   55
## [79,] 1.467218e-04   56
## [80,] 1.336874e-04   57
## [81,] 1.218110e-04   57
## [82,] 1.109896e-04   58
## [83,] 1.011296e-04   57
## [84,] 9.214556e-05   56
## [85,] 8.395960e-05   57
## [86,] 7.650086e-05   57
## [87,] 6.970473e-05   57
## [88,] 6.351236e-05   58
## [89,] 5.787010e-05   57
## [90,] 5.272908e-05   57
## [91,] 4.804477e-05   57
## [92,] 4.377660e-05   57
## [93,] 3.988761e-05   57
## [94,] 3.634411e-05   57
## [95,] 3.311540e-05   56
## [96,] 3.017352e-05   57
```

Nous observons encore ici, de très faibles valeurs de lambda. Celle qui minimise l'erreur, réduit de 40 variables le modèle. Lambda.1se qui régularise le plus le modèle, sélectionne "seulement" 17 variables.

Comparons enfin les prédictions selon les valeurs de lambda.

```
#prédiction avec lambda.min qui minimise l'erreur
pred_elet_min <- predict(modele_0.5, X_test, s=c(lambda.elet$lambda.min),
type="class")
```

```r
mc_elet_min <-  confusionMatrix(factor(pred_elet_min), factor(y_test_rl),
positive = "1")

cat("###################################\n")

## ###################################

cat("Matrice de confusion pour lambda.min \n")

## Matrice de confusion pour lambda.min

cat("###################################\n")

## ###################################

print(mc_elet_min)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3137  484
##          1  143  423
##
##                Accuracy : 0.8503
##                  95% CI : (0.8391, 0.8609)
##     No Information Rate : 0.7834
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4893
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.4664
##             Specificity : 0.9564
##          Pos Pred Value : 0.7473
##          Neg Pred Value : 0.8663
##              Prevalence : 0.2166
##          Detection Rate : 0.1010
##    Detection Prevalence : 0.1352
##       Balanced Accuracy : 0.7114
##
##        'Positive' Class : 1
##

cat("\n")

#Prédiction avec lambda.1se qui est le modèle le plus régularisé
pred_elet_1se <- predict(modele_0.5, X_test, s=c(lambda.elet$lambda.1se),
type="class")
mc_elet_1se <-  confusionMatrix(data = factor(pred_elet_1se), reference =
factor(y_test_rl), positive = "1")
```

```
cat("###################################\n")

## ###################################

cat("Matrice de confusion pour lambda.1se \n")

## Matrice de confusion pour lambda.1se

cat("###################################\n")

## ###################################

print(mc_elet_1se)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3177  534
##          1  103  373
##
##                Accuracy : 0.8479
##                  95% CI : (0.8366, 0.8586)
##     No Information Rate : 0.7834
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4587
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.41125
##             Specificity : 0.96860
##          Pos Pred Value : 0.78361
##          Neg Pred Value : 0.85610
##              Prevalence : 0.21662
##          Detection Rate : 0.08909
##    Detection Prevalence : 0.11369
##       Balanced Accuracy : 0.68992
##
##        'Positive' Class : 1
##
```

Sans surprise, c'est encore lorsque l'erreur est minimisée que l'on obtient les meilleurs résultats. Ainsi la courbe ROC avec lambda.min donne

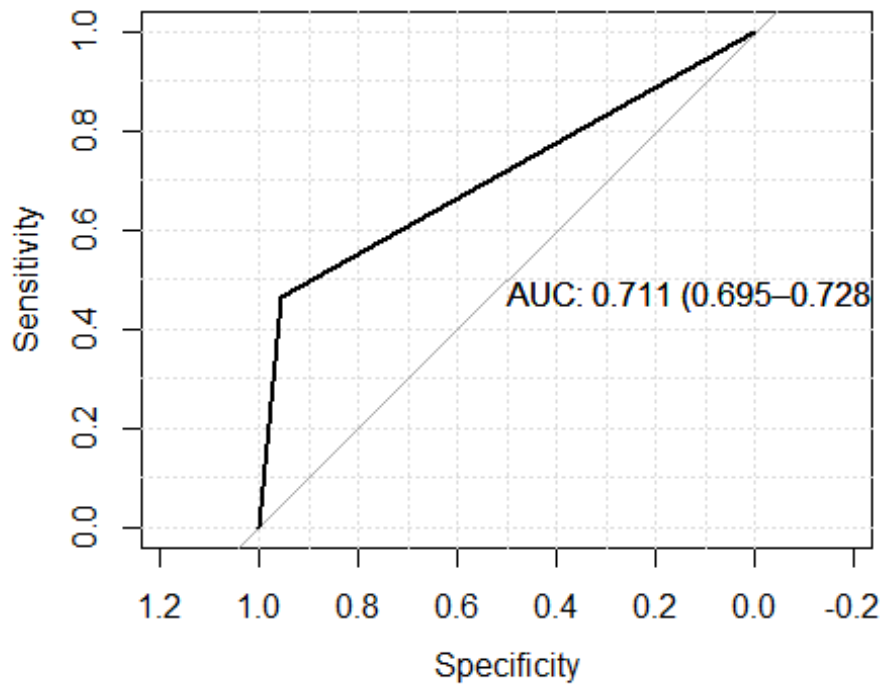
```
#Coeff du modèle
print(coef(lambda.elet,s="lambda.min"))

## 61 x 1 sparse Matrix of class "dgCMatrix"
##                                    s1
## (Intercept)           -1.937409767
```

```
## Rainfall               0.025132434
## WindGustSpeed          0.726417436
## WindSpeed9am           0.003775276
## WindSpeed3pm          -0.234258236
## Humidity9am            0.335535766
## Humidity3pm            1.559055782
## Pressure9am            .
## Pressure3pm           -0.168077459
## Temp9am                .
## Temp3pm                .
## Location_AliceSprings  .
## Location_Cairns       -0.445211258
## Location_Perth         .
## Location_Sydney       -0.498096797
## WindGustDir_E          .
## WindGustDir_ENE       -0.208178229
## WindGustDir_ESE        .
## WindGustDir_N         -0.013614783
## WindGustDir_NE        -0.018517951
## WindGustDir_NNE       -0.206834658
## WindGustDir_NNW        0.244863069
## WindGustDir_NW         0.272428740
## WindGustDir_S          .
## WindGustDir_SE         0.055522839
## WindGustDir_SSE        .
## WindGustDir_SSW        .
## WindGustDir_SW        -0.346720277
## WindGustDir_W          .
## WindGustDir_WSW        0.267566881
## WindDir9am_E           .
## WindDir9am_ENE         0.074178006
## WindDir9am_ESE        -0.131650520
## WindDir9am_N           0.606656888
## WindDir9am_NE          0.255171070
## WindDir9am_NNE         0.280736141
## WindDir9am_NNW         .
## WindDir9am_NW          0.067032834
## WindDir9am_S           .
## WindDir9am_SE         -0.036121827
## WindDir9am_SSE        -0.069604092
## WindDir9am_SW          .
## WindDir9am_W           0.109175440
## WindDir9am_WNW         .
## WindDir9am_WSW         .
## WindDir3pm_E           .
## WindDir3pm_ENE        -0.116973974
## WindDir3pm_ESE         0.023142424
## WindDir3pm_N           0.066566686
## WindDir3pm_NE         -0.284838049
## WindDir3pm_NNE         .
```

```
## WindDir3pm_NNW          0.818473163
## WindDir3pm_NW           0.525470034
## WindDir3pm_S            .
## WindDir3pm_SE           0.112354445
## WindDir3pm_SSE          .
## WindDir3pm_SSW         -0.391234937
## WindDir3pm_SW          -0.700299224
## WindDir3pm_WNW          0.191669574
## WindDir3pm_WSW         -0.147736207
## RainToday_Yes           0.597536168
```

```
# Courbe ROC
pred_elet_min <- as.integer(pred_elet_min)
ROC_elet <- roc(y_test_rl,pred_elet_min,
            ci=TRUE, ci.alpha=0.9, stratified=FALSE,
            plot=TRUE, grid=TRUE,
            print.auc=TRUE)
```

```
## Warning in roc.default(y_test_rl, pred_elet_min, ci = TRUE, ci.alpha =
0.9, :
## Deprecated use a matrix as response. Unexpected results may be produced,
please
## pass a vector or factor.
```



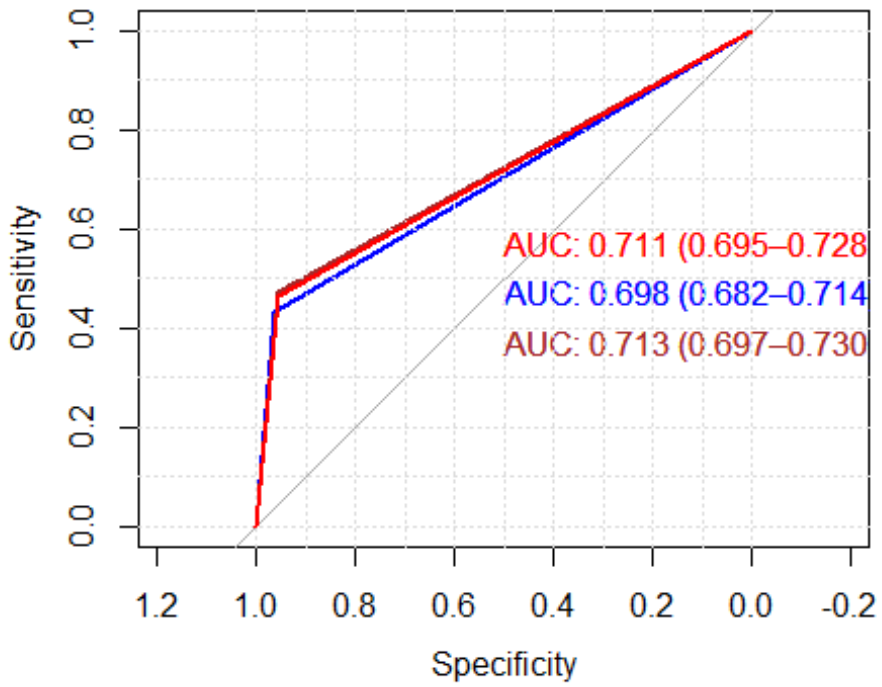

```
print(ROC_elet)
```

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

```
## roc.default(response = y_test_rl, predictor = pred_elet_min,     ci =
TRUE, plot = TRUE, ci.alpha = 0.9, stratified = FALSE,     grid = TRUE,
print.auc = TRUE)
##
## Data: pred_elet_min in 3280 controls (y_test_rl 0) < 907 cases (y_test_rl
1).
## Area under the curve: 0.7114
## 95% CI: 0.6948-0.728 (DeLong)
```

Comparaison des modèles

```
pred_ridge_min <- as.integer(pred_ridge_min)
ROC_ridge <- roc(y_test_rl,pred_ridge_min,
            ci=TRUE, ci.alpha=0.9, stratified=FALSE,
            plot=TRUE, grid=TRUE,
            print.auc=TRUE, col = 'blue')

## Warning in roc.default(y_test_rl, pred_ridge_min, ci = TRUE, ci.alpha =
0.9, :
## Deprecated use a matrix as response. Unexpected results may be produced,
please
## pass a vector or factor.

ROC_lasso <- roc(y_test_rl,pred_lasso_min,
            ci=TRUE, ci.alpha=0.9, stratified=FALSE,
            plot=TRUE, grid=TRUE,
            print.auc=TRUE, print.auc.y=0.4, add = TRUE, col = 'brown')

## Warning in roc.default(y_test_rl, pred_lasso_min, ci = TRUE, ci.alpha =
0.9, :
## Deprecated use a matrix as response. Unexpected results may be produced,
please
## pass a vector or factor.

ROC_elet <- roc(y_test_rl,pred_elet_min,
            ci=TRUE, ci.alpha=0.9, stratified=FALSE,
            plot=TRUE, grid=TRUE,
            print.auc=TRUE, print.auc.y=0.6, add = TRUE, col = 'red')

## Warning in roc.default(y_test_rl, pred_elet_min, ci = TRUE, ci.alpha =
0.9, :
## Deprecated use a matrix as response. Unexpected results may be produced,
please
## pass a vector or factor.
```



Conclusion : de manière globale, les performances sont assez proches entre ridge, lasso et la combinaison des deux. Cependant, c'est lorsque l'on penche vers lasso que les résultats sont meilleures. Une meilleure accuracy, mais surtout un meilleur rappel. Pour chaque cas, c'est la valeur de lambda qui minimise l'erreur qui est conservée. Mais on a pu se rendre compte de leurs très faibles valeurs, ce qui signifie que l'ont donne peu d'importance à la régularisation. Cela se voit par le faible nombre de variables nullifiées avec lasso.

En ElasticNet entre alpha = 0.2 et alpha = 0.5, on observe peu de différences. On retrouve cette similitude entre une pénalisation lasso et elasticnet (alpha <= 0) au niveau des courbes ROC dont les aires sont équivalentes. On sélectionnera le modèle Elastic Net qui a l'avantage de ne pas être biaisé par la corrélation entre variables lors de la sélection.

Comparaison des courbes ROC de tous les modèles :

```
ROC_elet <- roc(y_test_rl,pred_elet_min,
            ci=TRUE, ci.alpha=0.9, stratified=FALSE,
            plot=TRUE, grid=TRUE,
            print.auc=TRUE, col = 'brown')

## Warning in roc.default(y_test_rl, pred_elet_min, ci = TRUE, ci.alpha =
0.9, :
## Deprecated use a matrix as response. Unexpected results may be produced,
please
## pass a vector or factor.

ROC_svm_poly <- roc(y_test_final,pred_poly,
            ci=TRUE, ci.alpha=0.9, stratified=FALSE,
```

```
            plot=TRUE, grid=TRUE,
            print.auc=TRUE, print.auc.y=0.4, col = 'red', add = TRUE)

## Warning in roc.default(y_test_final, pred_poly, ci = TRUE, ci.alpha = 0.9,
:
## Deprecated use a matrix as response. Unexpected results may be produced,
please
## pass a vector or factor.

ROC_rnn_logistic <- roc(results$actual,results$prediction,
            ci=TRUE, ci.alpha=0.9, stratified=FALSE,
            plot=TRUE, grid=TRUE,
            print.auc=TRUE, print.auc.y=0.6, col = 'blue', add = TRUE)
```