TP n°2 - Classification: régression logistique et SVM

Salim Nadir et Guillaume Ostrom

Application I : Régression logistique

Nous constatons que le fichier *SaHeart.info* décrit les différentes variables présentes dans le fichier *SaHeart.txt*, où la dernière colonne (chd) est la variable cible.

1. Les données

Procédons au chargement du fichier *SaHeart.txt* :

```
setwd(getwd())
heartData = read.table('SaHeart.txt',header=TRUE,sep=",")
summary(heartData)
```

```
##    row.names          sbp           tobacco             ldl        
##  Min.   :  1.0   Min.   :101.0   Min.   : 0.0000   Min.   : 0.980  
##  1st Qu.:116.2   1st Qu.:124.0   1st Qu.: 0.0525   1st Qu.: 3.283  
##  Median :231.5   Median :134.0   Median : 2.0000   Median : 4.340  
##  Mean   :231.9   Mean   :138.3   Mean   : 3.6356   Mean   : 4.740  
##  3rd Qu.:347.8   3rd Qu.:148.0   3rd Qu.: 5.5000   3rd Qu.: 5.790  
##  Max.   :463.0   Max.   :218.0   Max.   :31.2000   Max.   :15.330  
##    adiposity        famhist        typea          obesity     
##  Min.   : 6.74   Absent :270   Min.   :13.0   Min.   :14.70  
##  1st Qu.:19.77   Present:192   1st Qu.:47.0   1st Qu.:22.98  
##  Median :26.11                 Median :53.0   Median :25.80  
##  Mean   :25.41                 Mean   :53.1   Mean   :26.04  
##  3rd Qu.:31.23                 3rd Qu.:60.0   3rd Qu.:28.50  
##  Max.   :42.49                 Max.   :78.0   Max.   :46.58  
##     alcohol            age              chd        
##  Min.   :  0.00   Min.   :15.00   Min.   :0.0000  
##  1st Qu.:  0.51   1st Qu.:31.00   1st Qu.:0.0000  
##  Median :  7.51   Median :45.00   Median :0.0000  
##  Mean   : 17.04   Mean   :42.82   Mean   :0.3463  
##  3rd Qu.: 23.89   3rd Qu.:55.00   3rd Qu.:1.0000  
##  Max.   :147.19   Max.   :64.00   Max.   :1.0000
```

```
attributes(heartData)$names
```

```
##  [1] "row.names" "sbp"       "tobacco"   "ldl"       "adiposity"
##  [6] "famhist"   "typea"     "obesity"   "alcohol"   "age"      
## [11] "chd"
```

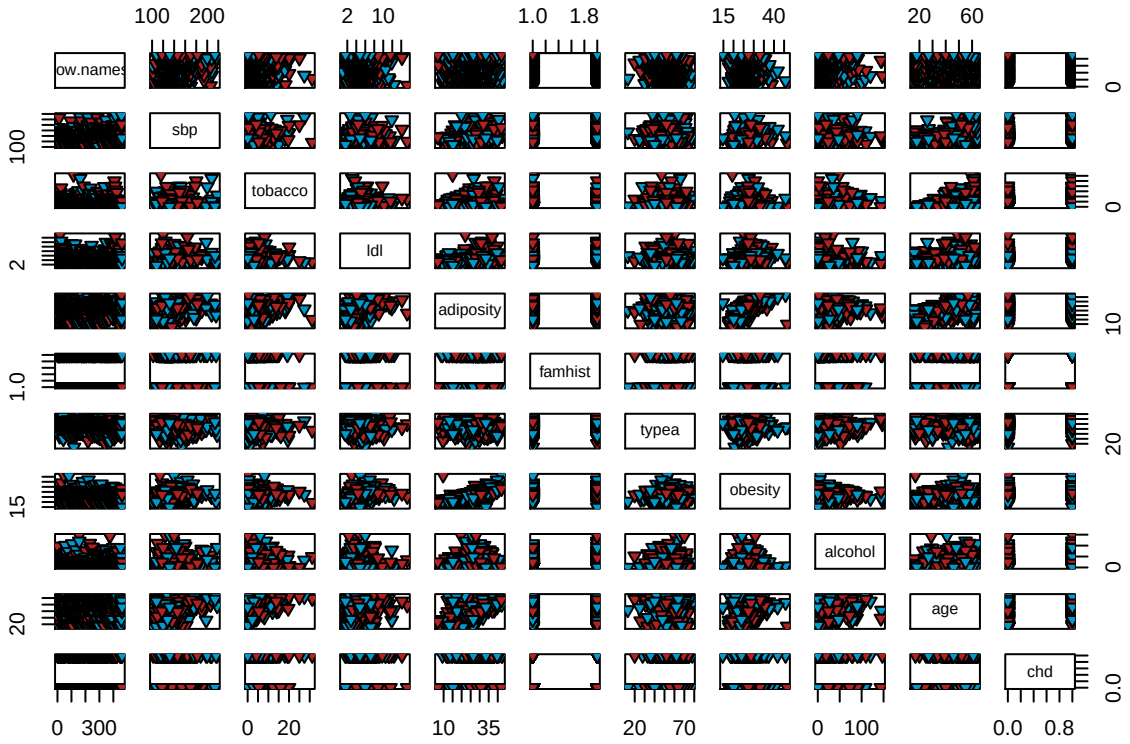
Voici les correspondances de chaque variable:
- sbp : la pression systolique,

- tobacco : la quantité en kilogrammes de tabac ingérée,
- ldl : lipoprotéine de basse densité,
- famhist : antécédent d'attaque cardiaque,
- typea : type-A,
- obesity : indice de masse corporelle,
- alcohol : consommation d'alcool par semaine en litre,
- age : âge de l'individu,
- chd : attaque cardiaque observée chez l'individu.

2. Scatterplot

Affichons le scatterplot du jeu de données:

```
pairs(heartData, pch=25, bg=c("firebrick","deepskyblue3"))[unclass(factor(heartData[, "chd"]))]
```




```
## NULL
```

3. Régression logistique

```
regLogistic = glm("chd~.", family=gaussian, heartData)
summary(regLogistic)
```

```
##
## Call:
## glm(formula = "chd~.", family = gaussian, data = heartData)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -0.7635  -0.3319  -0.1058   0.3742   1.0376
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -0.4673037  0.2080607  -2.246 0.025188 *
## row.names      -0.0001633  0.0001524  -1.071 0.284568
## sbp             0.0015503  0.0010762   1.441 0.150411
## tobacco         0.0162667  0.0048692   3.341 0.000905 ***
## ldl             0.0324112  0.0106970   3.030 0.002587 **
## adiposity       0.0027633  0.0047882   0.577 0.564147
## famhistPresent  0.1750802  0.0412966   4.240 2.72e-05 ***
## typea           0.0057758  0.0020565   2.809 0.005193 **
## obesity        -0.0117786  0.0070532  -1.670 0.095618 .
## alcohol        -0.0001667  0.0008308  -0.201 0.841034
## age             0.0066377  0.0019958   3.326 0.000954 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.1767215)
##
##     Null deviance: 104.589  on 461  degrees of freedom
## Residual deviance:  79.701  on 451  degrees of freedom
## AIC: 523.24
##
## Number of Fisher Scoring iterations: 2
```

On constate que les $p-values$ des variables tabacco, famhist et age sont faibles donc fortement significatives. Nous observons que les variables tabacco, famhist et age sont les variables les plus explicatives.

4. Comparaisons

On calcule la matrice de confusion.

```
predictHeartAttack = predict(regLogistic, heartData)

getConfusionMatrix = function(predictHeartAttack, heartData) {

  heartDataSize = length(predictHeartAttack)
  predictHeartAttackBoolean = c()

  confMatrix = matrix( c(0,0,0,0), nrow=2, ncol=2, byrow = TRUE)
  dimnames(confMatrix) = list( c("pred=0", "pred=1"),c("chd=0", "chd=1"))

  for (i in 1:heartDataSize){
    if(predictHeartAttack[i]>0.5){
      predictHeartAttackBoolean[i] = 1
      if(heartData$chd[i]==1){
```

```
        confMatrix[2,2] = confMatrix[2,2] + 1
      }else{
         confMatrix[2,1] = confMatrix[2,1] + 1
      }
    }else{
      predictHeartAttackBoolean[i] = 0
      if(heartData$chd[i]==1){
        confMatrix[1,2] = confMatrix[1,2] + 1
      }else{
         confMatrix[1,1] = confMatrix[1,1] + 1
      }
    }
  }

probaFaslePositive = confMatrix[2,1]/(confMatrix[1,1] + confMatrix[2,1])
probaFasleNegative = confMatrix[1,2]/(confMatrix[1,2] + confMatrix[2,2])

  return (list ("Matrice de confusion"=confMatrix, "Probabilité de risque de faux positif "=probaFasleP
}

confusionMatrix = getConfusionMatrix(predictHeartAttack, heartData)
print(confusionMatrix)
```

```
## $`Matrice de confusion`
##        chd=0 chd=1
## pred=0   262    78
## pred=1    40    82
##
## $`Probabilité de risque de faux positif `
## [1] 0.1324503
##
## $`Probabilité de risque de faux négatif `
## [1] 0.4875
```

```
#NB: Methode de calcul plus directe de la matrice de confusion:
matrixConf2 = table(regLogistic$fitted.values>0.5,heartData[,"chd"])
print(matrixConf2)
```

```
##
##           0   1
##   FALSE 262  78
##   TRUE   40  82
```

On constate 13 % de risque de faux positif et 49 % de risque de faux négatif.

5. Validation croisée

Nous définisons 75% de nos données comme étant des données d'apprentissage, les 25% restant seront destinées à tester notre modèle.

```r
#On prend aléatoirement sans remise les données d'apprentissage (75%) et les données de test (25%).
dt = sort(sample(nrow(heartData), 0.75*nrow(heartData)))
trainData75<-heartData[dt,]
testData25<-heartData[-dt,]

#Regression logistic sur les données d'apprentissage
regLogistic75 = glm("chd~.", family=gaussian, trainData75)
summary(regLogistic75)
```

```
##
## Call:
## glm(formula = "chd~.", family = gaussian, data = trainData75)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -0.89800  -0.33098  -0.07637   0.36471   1.04986
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -0.5421556  0.2319968  -2.337 0.020033 *
## row.names      -0.0001279  0.0001748  -0.732 0.464704
## sbp             0.0013768  0.0012285   1.121 0.263215
## tobacco         0.0152158  0.0057580   2.643 0.008614 **
## ldl             0.0389489  0.0124599   3.126 0.001927 **
## adiposity       0.0004083  0.0054316   0.075 0.940126
## famhistPresent  0.1450772  0.0477514   3.038 0.002567 **
## typea           0.0066049  0.0023370   2.826 0.004992 **
## obesity        -0.0113160  0.0076823  -1.473 0.141692
## alcohol         0.0003982  0.0009662   0.412 0.680532
## age             0.0088431  0.0024076   3.673 0.000279 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.1759232)
##
##     Null deviance: 80.116  on 345  degrees of freedom
## Residual deviance: 58.934  on 335  degrees of freedom
## AIC: 393.48
##
## Number of Fisher Scoring iterations: 2
```

```r
#Prediction avec les données de test
predictHeartAttack75_25 = predict(regLogistic75, testData25)
#Matrice de confusion du modèle
confusionMatrix75_25 = getConfusionMatrix(predictHeartAttack75_25, testData25)
print(confusionMatrix75_25)
```

```
## $`Matrice de confusion`
##        chd=0 chd=1
## pred=0    64    13
## pred=1    18    21
##
```

```
## $`Probabilité de risque de faux positif `
## [1] 0.2195122
##
## $`Probabilité de risque de faux négatif `
## [1] 0.3823529
```

Pour ce modèle de *cross validation* nous obtenons un risque de faux positif de 13.2% et un risque de faux négatif de 50.0%. Notons que ces résultats sont proches d'une regression logistique sur l'ensemble des données, on peut en déduire la bonne significativité du modèle.

Répétons plusieurs fois cette procédure.

```
checkCrossValidation = function(heartData, iterations) {

 confMatrix75_25 = c()
 error = c()
    for (i in 1:iterations) {

      dt = sort(sample(nrow(heartData), 0.75*nrow(heartData)))
      data75<-heartData[dt,]
      data25<-heartData[-dt,]

      #Regression logistic sur les donn?es d'apprentissage
      regLog75 = glm("chd~.", family=gaussian, data75)
      #Prediction avec les donn?es de test
      predict75_25 = predict(regLog75, data25)
      #Matrice de confusion du mod?le
      confMat = getConfusionMatrix(predict75_25, data25)
      confMatrix75_25[i] = confMat[1]
      error[i] = (as.numeric(confMat[[1]][1,2])+as.numeric(confMat[[1]][2,1]) )/  (as.numeric(confMat[[

    }

 return (list("Erreur minimum"=error[which.min(error)], "Erreur maximum"=error[which.max(error)],"Erreu
}
print(checkCrossValidation(heartData, 250))
```

```
## $`Erreur minimum`
## [1] 0.1724138
##
## $`Erreur maximum`
## [1] 0.362069
##
## $`Erreur moyenne`
## [1] 0.2706552
```

On remarque que l'erreur minimum est à 17% et que l'erreur maximum est élevée à 38% avec une moyenne d'erreur à 27%, ce qui est acceptable.
Tester son modéle sur les données d'entrainement conduirait à sous-estimer l'erreur, et donc a un biais. L'intérêt de cette approche est de tester la significativité d'un modèle entrainé sur 75% des données et de le tester sur les 25% données restantes.
On peut en déduire que notre modèle est plutôt pertinant.

6. Sélection des variables

```
# Backward
regBackward = step(regLogistic75, direction='backward', k=log(nrow(heartData)))
```

```
## Start:  AIC=438.97
## chd ~ row.names + sbp + tobacco + ldl + adiposity + famhist +
##     typea + obesity + alcohol + age
##
##             Df Deviance    AIC
## - adiposity  1   58.935 432.84
## - alcohol    1   58.964 433.01
## - row.names  1   59.029 433.39
## - sbp        1   59.155 434.13
## - obesity    1   59.316 435.07
## <none>           58.934 438.97
## - tobacco    1   60.163 439.97
## - typea      1   60.340 440.99
## - famhist    1   60.558 442.24
## - ldl        1   60.653 442.78
## - age        1   61.308 446.50
##
## Step:  AIC=432.84
## chd ~ row.names + sbp + tobacco + ldl + famhist + typea + obesity +
##     alcohol + age
##
##             Df Deviance    AIC
## - alcohol    1   58.965 426.88
## - row.names  1   59.029 427.25
## - sbp        1   59.160 428.02
## - obesity    1   59.585 430.50
## <none>           58.935 432.84
## - tobacco    1   60.163 433.84
## - typea      1   60.340 434.85
## - famhist    1   60.558 436.11
## - ldl        1   60.756 437.23
## - age        1   62.288 445.85
##
## Step:  AIC=426.88
## chd ~ row.names + sbp + tobacco + ldl + famhist + typea + obesity +
##     age
##
##             Df Deviance    AIC
## - row.names  1   59.051 421.25
## - sbp        1   59.207 422.16
## - obesity    1   59.604 424.48
## <none>           58.965 426.88
## - tobacco    1   60.262 428.27
## - typea      1   60.402 429.08
## - famhist    1   60.616 430.30
## - ldl        1   60.756 431.10
## - age        1   62.339 440.00
##
```

```
## Step:  AIC=421.25
## chd ~ sbp + tobacco + ldl + famhist + typea + obesity + age
##
##           Df Deviance    AIC
## - sbp      1   59.246 416.25
## - obesity  1   59.678 418.77
## <none>         59.051 421.25
## - tobacco  1   60.371 422.76
## - typea    1   60.642 424.31
## - famhist  1   60.672 424.48
## - ldl      1   60.929 425.95
## - age      1   62.539 434.97
##
## Step:  AIC=416.25
## chd ~ tobacco + ldl + famhist + typea + obesity + age
##
##           Df Deviance    AIC
## - obesity  1   59.807 413.38
## <none>         59.246 416.25
## - tobacco  1   60.570 417.77
## - typea    1   60.871 419.48
## - famhist  1   60.877 419.51
## - ldl      1   61.171 421.18
## - age      1   63.534 434.29
##
## Step:  AIC=413.38
## chd ~ tobacco + ldl + famhist + typea + age
##
##           Df Deviance    AIC
## <none>         59.807 413.38
## - tobacco  1   61.106 414.68
## - typea    1   61.243 415.46
## - ldl      1   61.337 415.99
## - famhist  1   61.393 416.30
## - age      1   63.676 428.93
```

```
formula(regBackward)
```

```
## chd ~ tobacco + ldl + famhist + typea + age
## <environment: 0x000000001787b1c8>
```

```
#NB Forward
regForward = step(regLogistic75, direction='forward', k=log(nrow(heartData)))
```

```
## Start:  AIC=438.97
## chd ~ row.names + sbp + tobacco + ldl + adiposity + famhist +
##     typea + obesity + alcohol + age
```

```
formula(regForward)
```

```
## chd ~ row.names + sbp + tobacco + ldl + adiposity + famhist +
##     typea + obesity + alcohol + age
## <environment: 0x000000001787b1c8>
```

```r
# Both
regBoth = step(regLogistic75, direction='both', k=log(nrow(heartData)))
```

```
## Start:  AIC=438.97
## chd ~ row.names + sbp + tobacco + ldl + adiposity + famhist +
##     typea + obesity + alcohol + age
##
##             Df Deviance    AIC
## - adiposity  1   58.935 432.84
## - alcohol    1   58.964 433.01
## - row.names  1   59.029 433.39
## - sbp        1   59.155 434.13
## - obesity    1   59.316 435.07
## <none>           58.934 438.97
## - tobacco    1   60.163 439.97
## - typea      1   60.340 440.99
## - famhist    1   60.558 442.24
## - ldl        1   60.653 442.78
## - age        1   61.308 446.50
##
## Step:  AIC=432.84
## chd ~ row.names + sbp + tobacco + ldl + famhist + typea + obesity +
##     alcohol + age
##
##             Df Deviance    AIC
## - alcohol    1   58.965 426.88
## - row.names  1   59.029 427.25
## - sbp        1   59.160 428.02
## - obesity    1   59.585 430.50
## <none>           58.935 432.84
## - tobacco    1   60.163 433.84
## - typea      1   60.340 434.85
## - famhist    1   60.558 436.11
## - ldl        1   60.756 437.23
## + adiposity  1   58.934 438.97
## - age        1   62.288 445.85
##
## Step:  AIC=426.88
## chd ~ row.names + sbp + tobacco + ldl + famhist + typea + obesity +
##     age
##
##             Df Deviance    AIC
## - row.names  1   59.051 421.25
## - sbp        1   59.207 422.16
## - obesity    1   59.604 424.48
## <none>           58.965 426.88
## - tobacco    1   60.262 428.27
## - typea      1   60.402 429.08
## - famhist    1   60.616 430.30
## - ldl        1   60.756 431.10
## + alcohol    1   58.935 432.84
## + adiposity  1   58.964 433.01
## - age        1   62.339 440.00
```

```
##
## Step:  AIC=421.25
## chd ~ sbp + tobacco + ldl + famhist + typea + obesity + age
##
##             Df Deviance    AIC
## - sbp        1   59.246 416.25
## - obesity    1   59.678 418.77
## <none>           59.051 421.25
## - tobacco    1   60.371 422.76
## - typea      1   60.642 424.31
## - famhist    1   60.672 424.48
## - ldl        1   60.929 425.95
## + row.names  1   58.965 426.88
## + alcohol    1   59.029 427.25
## + adiposity  1   59.051 427.38
## - age        1   62.539 434.97
##
## Step:  AIC=416.25
## chd ~ tobacco + ldl + famhist + typea + obesity + age
##
##             Df Deviance    AIC
## - obesity    1   59.807 413.38
## <none>           59.246 416.25
## - tobacco    1   60.570 417.77
## - typea      1   60.871 419.48
## - famhist    1   60.877 419.51
## - ldl        1   61.171 421.18
## + sbp        1   59.051 421.25
## + alcohol    1   59.207 422.16
## + row.names  1   59.207 422.16
## + adiposity  1   59.243 422.37
## - age        1   63.534 434.29
##
## Step:  AIC=413.38
## chd ~ tobacco + ldl + famhist + typea + age
##
##             Df Deviance    AIC
## <none>           59.807 413.38
## - tobacco    1   61.106 414.68
## - typea      1   61.243 415.46
## - ldl        1   61.337 415.99
## + obesity    1   59.246 416.25
## - famhist    1   61.393 416.30
## + adiposity  1   59.589 418.25
## + sbp        1   59.678 418.77
## + row.names  1   59.770 419.30
## + alcohol    1   59.782 419.37
## - age        1   63.676 428.93
```

```
formula(regBoth)
```

```
## chd ~ tobacco + ldl + famhist + typea + age
## <environment: 0x000000001787b1c8>
```

Une régression en direction *backward* nous sélectionne les 4 variables suivantes:
- famhist
- age
- tabacco
- ldl

```
heartDataBackward = heartData[c(3, 4, 6, 10, 11)]
print(attributes(heartDataBackward)$names)
```

```
## [1] "tobacco" "ldl"     "famhist" "age"     "chd"
```

```
dtBackward = sort(sample(nrow(heartDataBackward), 0.75*nrow(heartDataBackward)))
dataBackward75<-heartDataBackward[dtBackward,]
dataBackward25<-heartDataBackward[-dtBackward,]

#Regression logistic sur les donn?es d'apprentissage
regLog75_Backward = glm("chd~.", family=gaussian, dataBackward75)
#Prediction avec les donn?es de test
predict75_25_Backward = predict(regLog75_Backward, dataBackward25)
#Matrice de confusion du mod?le
confMatBackward = getConfusionMatrix(predict75_25_Backward, dataBackward25)
print(confMatBackward)
```

```
## $`Matrice de confusion`
##        chd=0 chd=1
## pred=0    66    20
## pred=1    10    20
##
## $`Probabilité de risque de faux positif `
## [1] 0.1315789
##
## $`Probabilité de risque de faux négatif `
## [1] 0.5
```

```
print(checkCrossValidation(heartDataBackward, 150))
```

```
## $`Erreur minimum`
## [1] 0.1724138
##
## $`Erreur maximum`
## [1] 0.3706897
##
## $`Erreur moyenne`
## [1] 0.2777586
```

Nous observons qu'avec un step **backward** l'erreur minimum est à 18% et l'erreur maximum à 38% et la même erreur moyenne est à 27%.
Nous remarquons qu'il n'y a pas d'amélioration de l'erreur par rapport au modèle complet, ce qui donne un **avantage à ce modèle** car il nécessite moins de données pour le même résultat.

7. Courbe ROC

Comparons à l'aide des courbes ROC ces 3 modèles de régression logistique:
1)Modèle complet
2)Modèle avec les variables sélectionnées
3)Modèle avec la variable la plus significative

Import de la bibliothéque ROC

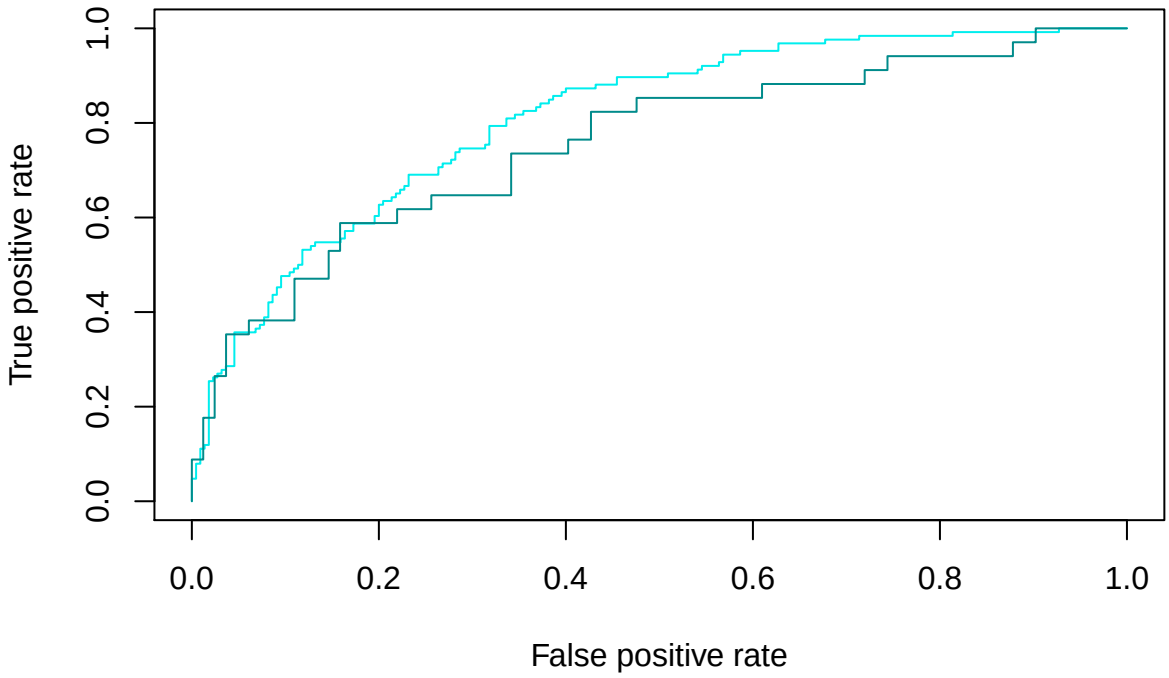
```
#install.packages("ROCR")
library(ROCR)
```

```
## Loading required package: gplots
```

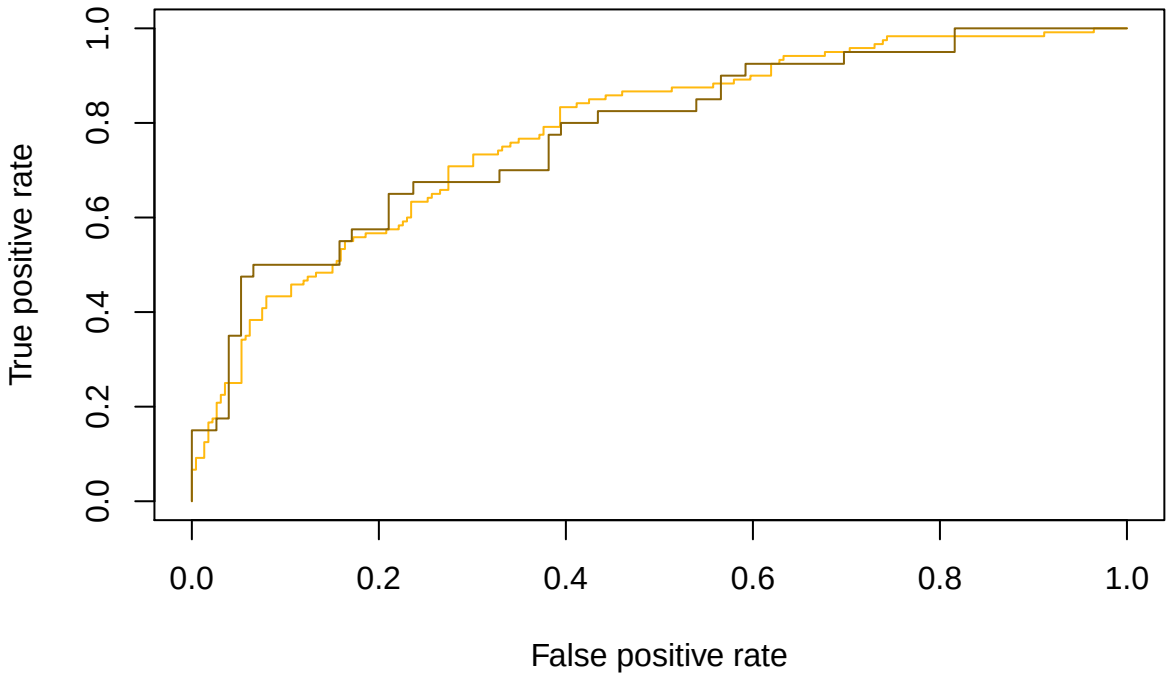
```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
##     lowess
```

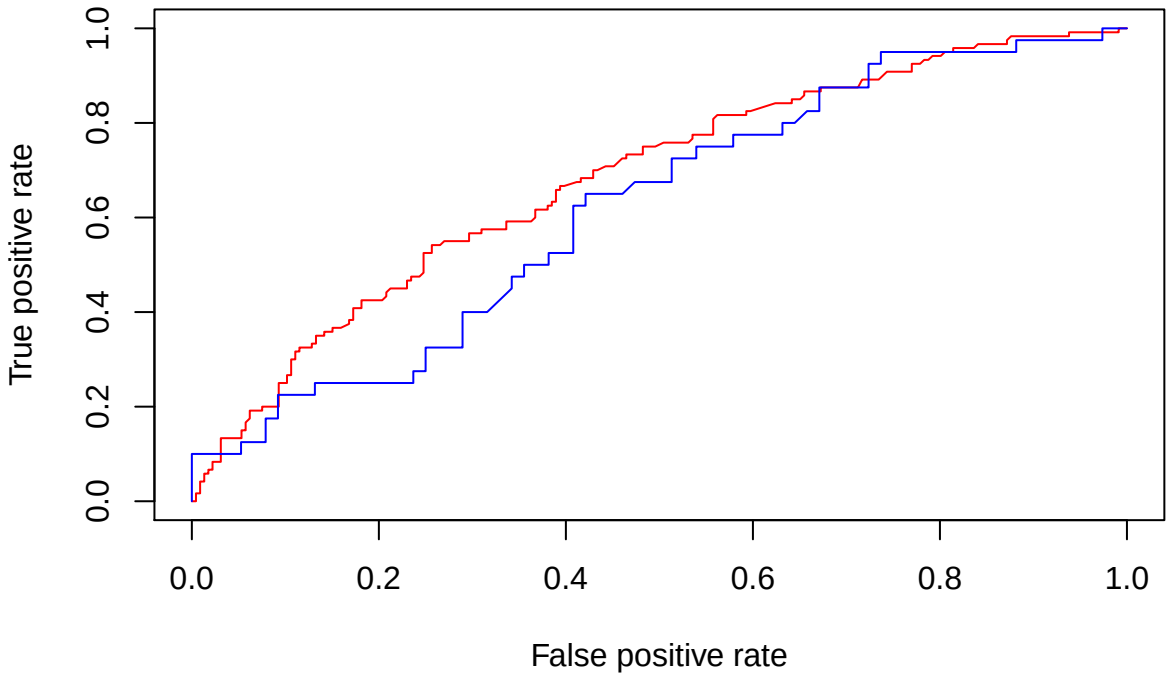
```
library(gplots)

plotRoc = function(predictTrain, dataTrainTest, color, added){
rocPred = prediction(predictTrain, dataTrainTest["chd"])
rocPerf = performance(rocPred, measure = "tpr", x.measure = "fpr")
plot(rocPerf, add= added, col=color)
}
#Mod?le Complet Train
predictHeartAttack75_75 = predict(regLogistic75, trainData75)
plotRoc(predictHeartAttack75_75,trainData75,"cyan2", FALSE)

#Mod?le Complet Test
plotRoc(predictHeartAttack75_25,testData25,"cyan4", TRUE)
```




```r
#Mod?le Backward Train
predict75_75_Backward = predict(regLog75_Backward, dataBackward75)
plotRoc(predict75_75_Backward,dataBackward75, "darkgoldenrod1",FALSE)

#Mod?le Backward Test
plotRoc(predict75_25_Backward,dataBackward25,"darkgoldenrod4",TRUE)
```




```
#Mod?le mono variable
dtmono = sort(sample(nrow(heartData), 0.75*nrow(heartData)))
dataMono75<-heartData[dtmono,]
dataMono25<-heartData[-dtmono,]

#Mod?le mono Train
regLogisticMono = glm("chd~ldl", family=gaussian, heartData)
predict75_75_mono = predict(regLogisticMono, dataMono75)
plotRoc(predict75_75_mono,dataMono75, "red",FALSE)

#Mod?le mono variable
predict75_25_mono = predict(regLogisticMono, dataMono25)
plotRoc(predict75_25_mono,dataMono25, "blue",TRUE)
```



En conclusion, le modèle complet n°1 donne certes des résultats comparable à celui du n°2, mais comporte plus de variables et donc de données pour y arriver. Aussi, le modèle n°3 qui se résume à une variable explicative n'est pas suffisament pertinant.
Donc les courbes ROC montrent que le **modèle n°2** issu d'une sélection des 4 meilleurs variables et le modèle le plus adapté.

Application II : Classification par SVM

Analyse préliminaire

1. Etude rapide

Toutes les informations liées aux données spam sont dans le fichier **spaminfo.txt**

2. Chargement des données

Nous procédons au chargement des données spam.

```
spamData = read.table("spam.txt", header=TRUE, sep=';')
```

3. Observations

```
summary(spamData)
```

```
##       A.1               A.2               A.3               A.4          
##  Min.   :0.0000   Min.   : 0.000   Min.   :0.0000   Min.   : 0.00000  
##  1st Qu.:0.0000   1st Qu.: 0.000   1st Qu.:0.0000   1st Qu.: 0.00000  
##  Median :0.0000   Median : 0.000   Median :0.0000   Median : 0.00000  
##  Mean   :0.1046   Mean   : 0.213   Mean   :0.2807   Mean   : 0.06542  
##  3rd Qu.:0.0000   3rd Qu.: 0.000   3rd Qu.:0.4200   3rd Qu.: 0.00000  
##  Max.   :4.5400   Max.   :14.280   Max.   :5.1000   Max.   :42.81000  
##       A.5               A.6               A.7               A.8          
##  Min.   : 0.0000   Min.   :0.0000   Min.   :0.0000   Min.   : 0.0000  
##  1st Qu.: 0.0000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.: 0.0000  
##  Median : 0.0000   Median :0.0000   Median :0.0000   Median : 0.0000  
##  Mean   : 0.3122   Mean   :0.0959   Mean   :0.1142   Mean   : 0.1053  
##  3rd Qu.: 0.3800   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.: 0.0000  
##  Max.   :10.0000   Max.   :5.8800   Max.   :7.2700   Max.   :11.1100  
##       A.9               A.10              A.11              A.12         
##  Min.   :0.00000   Min.   : 0.0000   Min.   :0.00000   Min.   :0.0000  
##  1st Qu.:0.00000   1st Qu.: 0.0000   1st Qu.:0.00000   1st Qu.:0.0000  
##  Median :0.00000   Median : 0.0000   Median :0.00000   Median :0.1000  
##  Mean   :0.09007   Mean   : 0.2394   Mean   :0.05982   Mean   :0.5417  
##  3rd Qu.:0.00000   3rd Qu.: 0.1600   3rd Qu.:0.00000   3rd Qu.:0.8000  
##  Max.   :5.26000   Max.   :18.1800   Max.   :2.61000   Max.   :9.6700  
##       A.13              A.14              A.15              A.16         
##  Min.   :0.00000   Min.   : 0.00000   Min.   :0.0000   Min.   : 0.0000  
##  1st Qu.:0.00000   1st Qu.: 0.00000   1st Qu.:0.0000   1st Qu.: 0.0000  
##  Median :0.00000   Median : 0.00000   Median :0.0000   Median : 0.0000  
##  Mean   :0.09393   Mean   : 0.05863   Mean   :0.0492   Mean   : 0.2488  
##  3rd Qu.:0.00000   3rd Qu.: 0.00000   3rd Qu.:0.0000   3rd Qu.: 0.1000  
##  Max.   :5.55000   Max.   :10.00000   Max.   :4.4100   Max.   :20.0000  
##       A.17              A.18              A.19              A.20         
##  Min.   :0.0000   Min.   :0.0000   Min.   : 0.000   Min.   : 0.00000  
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.: 0.000   1st Qu.: 0.00000  
##  Median :0.0000   Median :0.0000   Median : 1.310   Median : 0.00000  
##  Mean   :0.1426   Mean   :0.1847   Mean   : 1.662   Mean   : 0.08558  
##  3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.: 2.640   3rd Qu.: 0.00000  
##  Max.   :7.1400   Max.   :9.0900   Max.   :18.750   Max.   :18.18000  
##       A.21              A.22              A.23              A.24         
##  Min.   : 0.0000   Min.   : 0.0000   Min.   :0.0000   Min.   : 0.00000  
##  1st Qu.: 0.0000   1st Qu.: 0.0000   1st Qu.:0.0000   1st Qu.: 0.00000  
##  Median : 0.2200   Median : 0.0000   Median :0.0000   Median : 0.00000  
##  Mean   : 0.8098   Mean   : 0.1212   Mean   :0.1016   Mean   : 0.09427  
##  3rd Qu.: 1.2700   3rd Qu.: 0.0000   3rd Qu.:0.0000   3rd Qu.: 0.00000  
##  Max.   :11.1100   Max.   :17.1000   Max.   :5.4500   Max.   :12.50000  
##       A.25              A.26              A.27              A.28         
##  Min.   : 0.0000   Min.   : 0.0000   Min.   : 0.0000   Min.   :0.0000  
##  1st Qu.: 0.0000   1st Qu.: 0.0000   1st Qu.: 0.0000   1st Qu.:0.0000  
##  Median : 0.0000   Median : 0.0000   Median : 0.0000   Median :0.0000  
##  Mean   : 0.5495   Mean   : 0.2654   Mean   : 0.7673   Mean   :0.1248  
##  3rd Qu.: 0.0000   3rd Qu.: 0.0000   3rd Qu.: 0.0000   3rd Qu.:0.0000  
##  Max.   :20.8300   Max.   :16.6600   Max.   :33.3300   Max.   :9.0900
```

```
##       A.29               A.30              A.31               A.32
##  Min.   : 0.00000   Min.   :0.0000   Min.   : 0.00000   Min.   :0.00000
##  1st Qu.: 0.00000   1st Qu.:0.0000   1st Qu.: 0.00000   1st Qu.:0.00000
##  Median : 0.00000   Median :0.0000   Median : 0.00000   Median :0.00000
##  Mean   : 0.09892   Mean   :0.1029   Mean   : 0.06475   Mean   :0.04705
##  3rd Qu.: 0.00000   3rd Qu.:0.0000   3rd Qu.: 0.00000   3rd Qu.:0.00000
##  Max.   :14.28000   Max.   :5.8800   Max.   :12.50000   Max.   :4.76000
##       A.33               A.34              A.35               A.36
##  Min.   : 0.00000   Min.   :0.00000   Min.   : 0.0000   Min.   :0.00000
##  1st Qu.: 0.00000   1st Qu.:0.00000   1st Qu.: 0.0000   1st Qu.:0.00000
##  Median : 0.00000   Median :0.00000   Median : 0.0000   Median :0.00000
##  Mean   : 0.09723   Mean   :0.04784   Mean   : 0.1054   Mean   :0.09748
##  3rd Qu.: 0.00000   3rd Qu.:0.00000   3rd Qu.: 0.0000   3rd Qu.:0.00000
##  Max.   :18.18000   Max.   :4.76000   Max.   :20.0000   Max.   :7.69000
##       A.37             A.38              A.39               A.40
##  Min.   :0.000   Min.   :0.0000   Min.   : 0.00000   Min.   :0.00000
##  1st Qu.:0.000   1st Qu.:0.0000   1st Qu.: 0.00000   1st Qu.:0.00000
##  Median :0.000   Median :0.0000   Median : 0.00000   Median :0.00000
##  Mean   :0.137   Mean   :0.0132   Mean   : 0.07863   Mean   :0.06483
##  3rd Qu.:0.000   3rd Qu.:0.0000   3rd Qu.: 0.00000   3rd Qu.:0.00000
##  Max.   :6.890   Max.   :8.3300   Max.   :11.11000   Max.   :4.76000
##       A.41              A.42              A.43             A.44
##  Min.   :0.00000   Min.   : 0.0000   Min.   :0.0000   Min.   : 0.0000
##  1st Qu.:0.00000   1st Qu.: 0.0000   1st Qu.:0.0000   1st Qu.: 0.0000
##  Median :0.00000   Median : 0.0000   Median :0.0000   Median : 0.0000
##  Mean   :0.04367   Mean   : 0.1323   Mean   :0.0461   Mean   : 0.0792
##  3rd Qu.:0.00000   3rd Qu.: 0.0000   3rd Qu.:0.0000   3rd Qu.: 0.0000
##  Max.   :7.14000   Max.   :14.2800   Max.   :3.5700   Max.   :20.0000
##       A.45              A.46              A.47               A.48
##  Min.   : 0.0000   Min.   : 0.0000   Min.   :0.000000   Min.   : 0.00000
##  1st Qu.: 0.0000   1st Qu.: 0.0000   1st Qu.:0.000000   1st Qu.: 0.00000
##  Median : 0.0000   Median : 0.0000   Median :0.000000   Median : 0.00000
##  Mean   : 0.3012   Mean   : 0.1798   Mean   :0.005444   Mean   : 0.03187
##  3rd Qu.: 0.1100   3rd Qu.: 0.0000   3rd Qu.:0.000000   3rd Qu.: 0.00000
##  Max.   :21.4200   Max.   :22.0500   Max.   :2.170000   Max.   :10.00000
##       A.49              A.50             A.51              A.52
##  Min.   :0.00000   Min.   :0.000   Min.   :0.00000   Min.   : 0.0000
##  1st Qu.:0.00000   1st Qu.:0.000   1st Qu.:0.00000   1st Qu.: 0.0000
##  Median :0.00000   Median :0.065   Median :0.00000   Median : 0.0000
##  Mean   :0.03857   Mean   :0.139   Mean   :0.01698   Mean   : 0.2691
##  3rd Qu.:0.00000   3rd Qu.:0.188   3rd Qu.:0.00000   3rd Qu.: 0.3150
##  Max.   :4.38500   Max.   :9.752   Max.   :4.08100   Max.   :32.4780
##       A.53              A.54               A.55               A.56
##  Min.   :0.00000   Min.   : 0.00000   Min.   :   1.000   Min.   :   1.00
##  1st Qu.:0.00000   1st Qu.: 0.00000   1st Qu.:   1.588   1st Qu.:   6.00
##  Median :0.00000   Median : 0.00000   Median :   2.276   Median :  15.00
##  Mean   :0.07581   Mean   : 0.04424   Mean   :   5.191   Mean   :  52.17
##  3rd Qu.:0.05200   3rd Qu.: 0.00000   3rd Qu.:   3.706   3rd Qu.:  43.00
##  Max.   :6.00300   Max.   :19.82900   Max.   :1102.500   Max.   :9989.00
##       A.57            spam
##  Min.   :    1.0   email:2788
##  1st Qu.:   35.0   spam :1813
##  Median :   95.0
##  Mean   :  283.3
```

```
##  3rd Qu.:  266.0
##  Max.   :15841.0
```

Dans le fichier **spam.txt** nous observons 4601 observations, 58 variables dont 55 float, 2 int, 1 label et 1 boolean.

4. Head

```
head(spamData)
```

```
##    A.1  A.2  A.3 A.4  A.5  A.6  A.7  A.8  A.9 A.10 A.11 A.12 A.13 A.14
## 1 0.00 0.64 0.64   0 0.32 0.00 0.00 0.00 0.00 0.00 0.00 0.64 0.00 0.00
## 2 0.21 0.28 0.50   0 0.14 0.28 0.21 0.07 0.00 0.94 0.21 0.79 0.65 0.21
## 3 0.06 0.00 0.71   0 1.23 0.19 0.19 0.12 0.64 0.25 0.38 0.45 0.12 0.00
## 4 0.00 0.00 0.00   0 0.63 0.00 0.31 0.63 0.31 0.63 0.31 0.31 0.31 0.00
## 5 0.00 0.00 0.00   0 0.63 0.00 0.31 0.63 0.31 0.63 0.31 0.31 0.31 0.00
## 6 0.00 0.00 0.00   0 1.85 0.00 0.00 1.85 0.00 0.00 0.00 0.00 0.00 0.00
##   A.15 A.16 A.17 A.18 A.19 A.20 A.21 A.22 A.23 A.24 A.25 A.26 A.27 A.28
## 1 0.00 0.32 0.00 1.29 1.93 0.00 0.96    0 0.00 0.00    0    0    0    0
## 2 0.14 0.14 0.07 0.28 3.47 0.00 1.59    0 0.43 0.43    0    0    0    0
## 3 1.75 0.06 0.06 1.03 1.36 0.32 0.51    0 1.16 0.06    0    0    0    0
## 4 0.00 0.31 0.00 0.00 3.18 0.00 0.31    0 0.00 0.00    0    0    0    0
## 5 0.00 0.31 0.00 0.00 3.18 0.00 0.31    0 0.00 0.00    0    0    0    0
## 6 0.00 0.00 0.00 0.00 0.00 0.00 0.00    0 0.00 0.00    0    0    0    0
##   A.29 A.30 A.31 A.32 A.33 A.34 A.35 A.36 A.37 A.38 A.39 A.40 A.41 A.42
## 1    0    0    0    0    0    0    0    0 0.00    0    0 0.00    0    0
## 2    0    0    0    0    0    0    0    0 0.07    0    0 0.00    0    0
## 3    0    0    0    0    0    0    0    0 0.00    0    0 0.06    0    0
## 4    0    0    0    0    0    0    0    0 0.00    0    0 0.00    0    0
## 5    0    0    0    0    0    0    0    0 0.00    0    0 0.00    0    0
## 6    0    0    0    0    0    0    0    0 0.00    0    0 0.00    0    0
##   A.43 A.44 A.45 A.46 A.47 A.48 A.49  A.50 A.51  A.52  A.53  A.54  A.55
## 1 0.00    0 0.00 0.00    0    0 0.00 0.000    0 0.778 0.000 0.000 3.756
## 2 0.00    0 0.00 0.00    0    0 0.00 0.132    0 0.372 0.180 0.048 5.114
## 3 0.12    0 0.06 0.06    0    0 0.01 0.143    0 0.276 0.184 0.010 9.821
## 4 0.00    0 0.00 0.00    0    0 0.00 0.137    0 0.137 0.000 0.000 3.537
## 5 0.00    0 0.00 0.00    0    0 0.00 0.135    0 0.135 0.000 0.000 3.537
## 6 0.00    0 0.00 0.00    0    0 0.00 0.223    0 0.000 0.000 0.000 3.000
##   A.56 A.57 spam
## 1   61  278 spam
## 2  101 1028 spam
## 3  485 2259 spam
## 4   40  191 spam
## 5   40  191 spam
## 6   15   54 spam
```

Nous observons que la variable cible est située à la derniére colonne, la numéro 58.

5. Proportions

```
Y = spamData[,ncol(spamData)]
levels(Y)
```

```
## [1] "email" "spam"
```

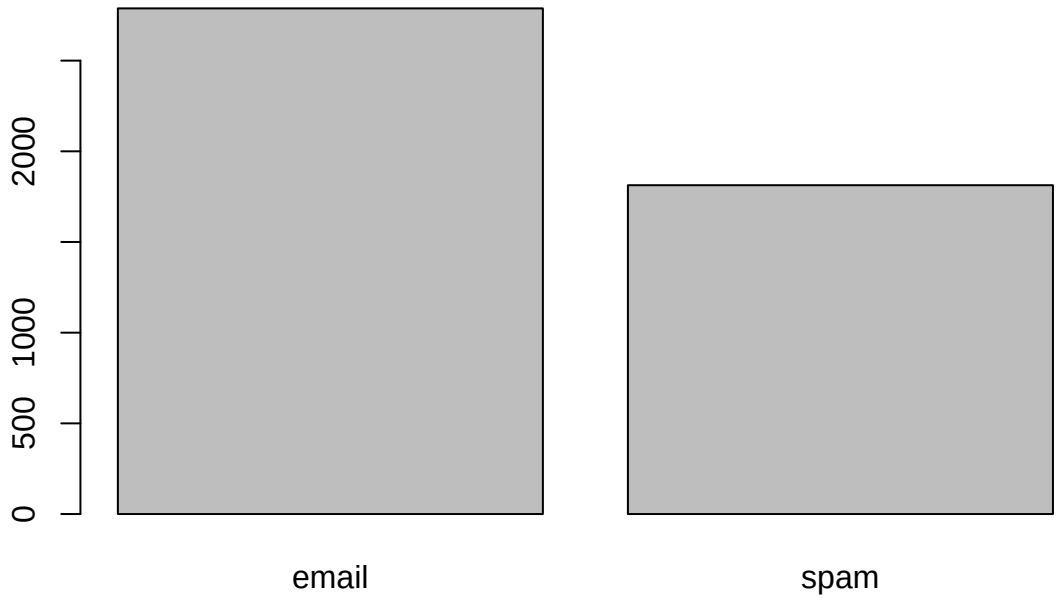
```
nlevels(Y)
```

```
## [1] 2
```

```
table(Y)
```

```
## Y
## email  spam
##  2788  1813
```

```
plot(Y)
```




```
spamProp = table(Y)[2]/(table(Y)[1]+table(Y)[2])
print(spamProp)
```

```
##      spam
## 0.3940448
```

```r
mailProp = 1 - spamProp
print(mailProp)
```

```
##      spam
## 0.6059552
```

La proportions de spam est de **39.4%** et la proportion de mail est de **60.6%**.

N.B. : La commande summary nous donne un résultat plus immédiat:

```r
summary(spamData$spam)
```

```
## email  spam
##  2788  1813
```

Classification par SVM

6. Données d'apprentissage et de test

Nous choisissons aléatoirement 75% de nos données comme étant des données d'apprentissage, les 25% restant seront destinées à tester notre modèle.

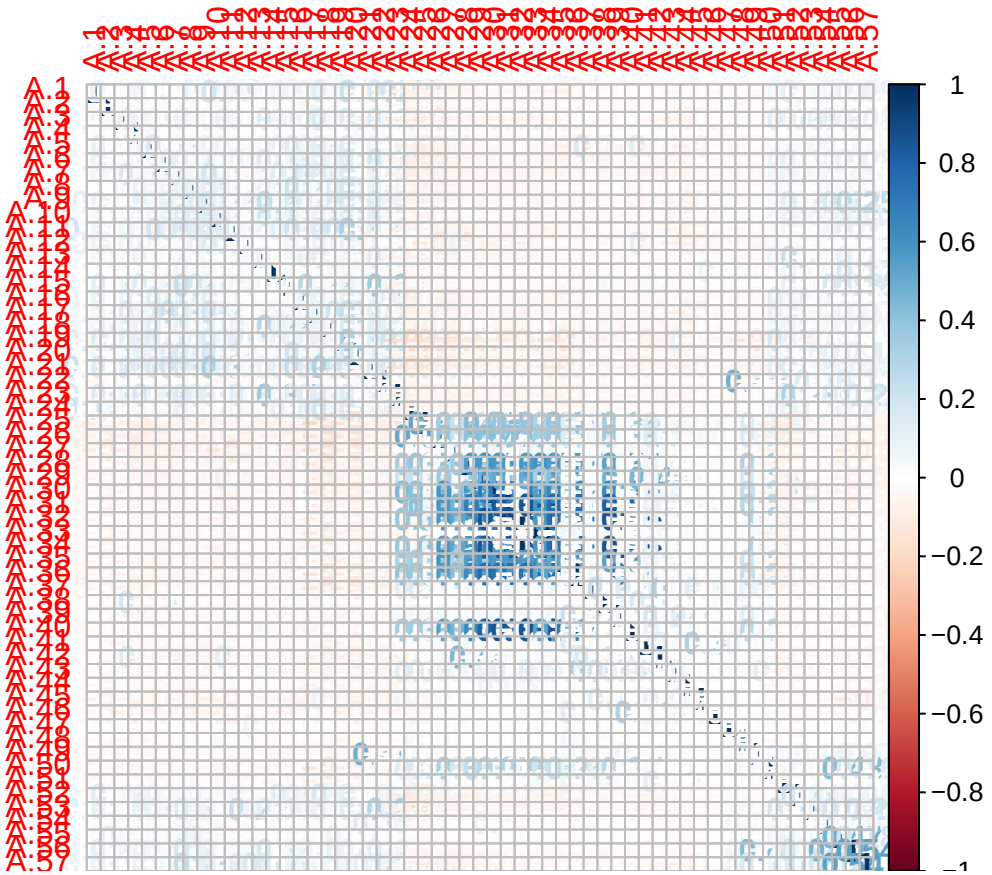
```r
#On prend al?atoirement sans remise les donn?es d'apprentissage (75%) et les donn?es de test (25%).
dtSpam = sort(sample(nrow(spamData), 0.75*nrow(spamData)))
trainSpamData75<-spamData[dtSpam,]
testSpamData25<-spamData[-dtSpam,]
```

Chargement des données.

```r
Xtrain = as.matrix(trainSpamData75[,-58])
Ytrain = as.matrix(trainSpamData75[,58])
```

Calculons les matrices de covariables

```r
library(corrplot)
corrplot(cor(trainSpamData75[,-58]),method="number")
```



7. Calibration C-SVM

Calibrons les données avec un noyau gaussien sur les données d'apprentissage :

```
#install.packages("kernlab")
library(kernlab)
# rbfdot Radial Basis kernel "Gaussian", polydot Polynomial kernel, vanilladot Linear kernel,tanhdot Hy

ksvmTrainSpam = ksvm(Xtrain, Ytrain, kernel="rbfdot", type="C-svc")
predictTrainSpam = predict(ksvmTrainSpam, Xtrain)
```

8. Caractéristiques de la base d'apprentissage

```
matrixConfSpam = table(predictTrainSpam, Ytrain)
print(matrixConfSpam)
```

```
##                 Ytrain
## predictTrainSpam email spam
##            email  2057  108
##            spam     50 1235
```

```r
FalsePositive = matrixConfSpam[1,2] / (matrixConfSpam[1,1]+matrixConfSpam[1,2])
print(FalsePositive)
```

```
## [1] 0.04988453
```

```r
FalseNegative = matrixConfSpam[2,1] / (matrixConfSpam[2,1]+matrixConfSpam[2,2])
print(FalseNegative)
```

```
## [1] 0.03891051
```

On note que les résultats son trés significatif avec moins de 5% d'erreur de faux positif et prés de 4% d'erreur de faux négatif.

9. Caractéristiques de la base de test

```r
Xtest = as.matrix(testSpamData25[,-58])
Ytest = as.matrix(testSpamData25[,58])
predictTestSpam = predict(ksvmTrainSpam, Xtest)
matrixConfSpamTest = table(predictTestSpam, Ytest)
print(matrixConfSpamTest)
```

```
##                Ytest
## predictTestSpam email spam
##           email   648   49
##           spam     33  421
```

```r
FalsePositive = matrixConfSpamTest[1,2] / (matrixConfSpamTest[1,1]+matrixConfSpamTest[1,2])
print(FalsePositive)
```

```
## [1] 0.07030129
```

```r
FalseNegative = matrixConfSpamTest[2,1] / (matrixConfSpamTest[2,1]+matrixConfSpamTest[2,2])
print(FalseNegative)
```

```
## [1] 0.07268722
```

On note que les résultats son trés significatif avec 56 d'erreur de faux positif et près de 5% d'erreur de faux négatif.

10. Mesure de l'impact du choix aléatoire de la base d'apprentissage

Mesurons pour 20 itérations l'erreur du modèle.

```r
getError = function(Noyau){
  # 1) Choix de la base
  dtSpam = sort(sample(nrow(spamData), 0.75*nrow(spamData)))
  trainSpamData75<-spamData[dtSpam,]
  testSpamData25<-spamData[-dtSpam,]

  # Initialisation des matrices
  Xtrain = as.matrix(trainSpamData75[,-58])
  Ytrain = as.matrix(trainSpamData75[,58])
  Xtest = as.matrix(testSpamData25[,-58])
  Ytest = as.matrix(testSpamData25[,58])

  # 2) Calibration du mod?le sur la base d'apprentissage
  ksvmTrainSpam = ksvm(Xtrain, Ytrain, kernel=Noyau, type="C-svc")

  # 3) Evaluation de l'erreur sur la base de test
  predictTestSpam = predict(ksvmTrainSpam, Xtest)
  matrixConfSpamTest = table(predictTestSpam, Ytest)

  # Retourne l'erreur
  return (as.numeric((matrixConfSpamTest[1,2]+matrixConfSpamTest[2,1]) / (matrixConfSpamTest[1,1]+matri
  }

error = c()
Noyau = "rbfdot"
for(k in 1:20){

  error[k] = getError(Noyau)

}
#Erreur minimum :
print(error[which.min(error)])
```

```
## [1] 0.05821025
```

```r
# Erreur maximum :
print(error[which.max(error)])
```

```
## [1] 0.08427454
```

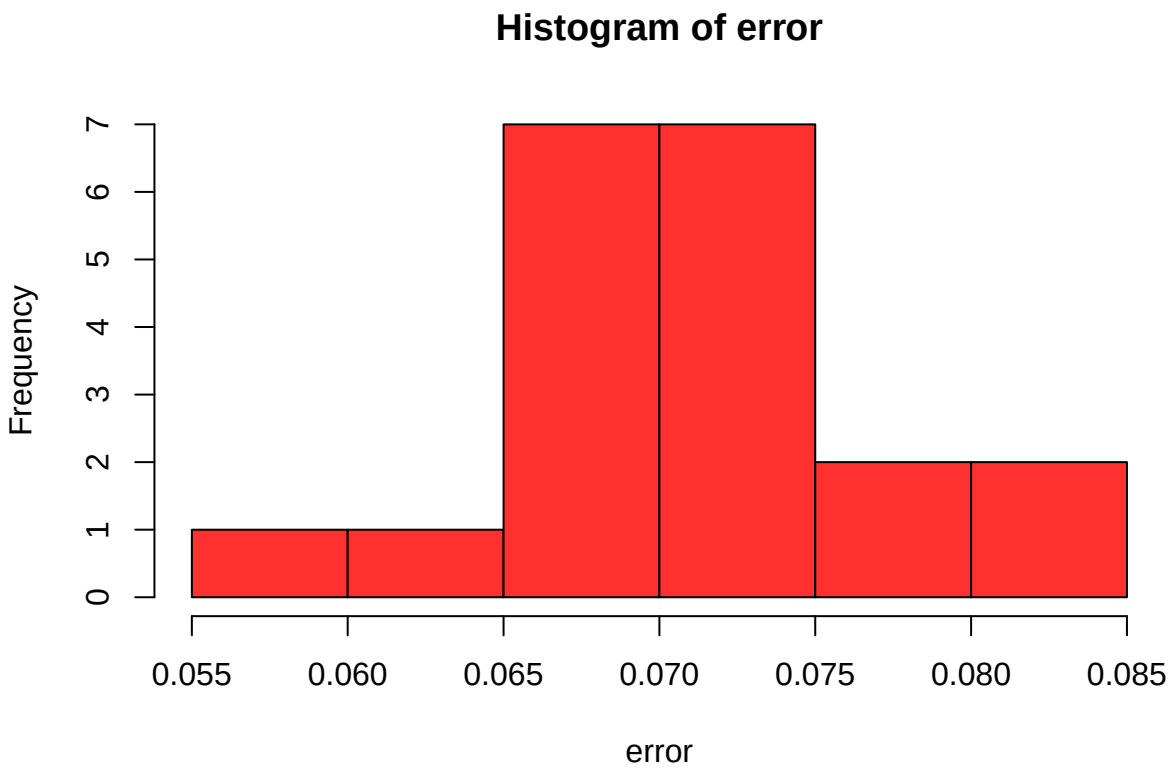
```r
# Erreur moyenne :
print(mean(error))
```

```
## [1] 0.07098175
```

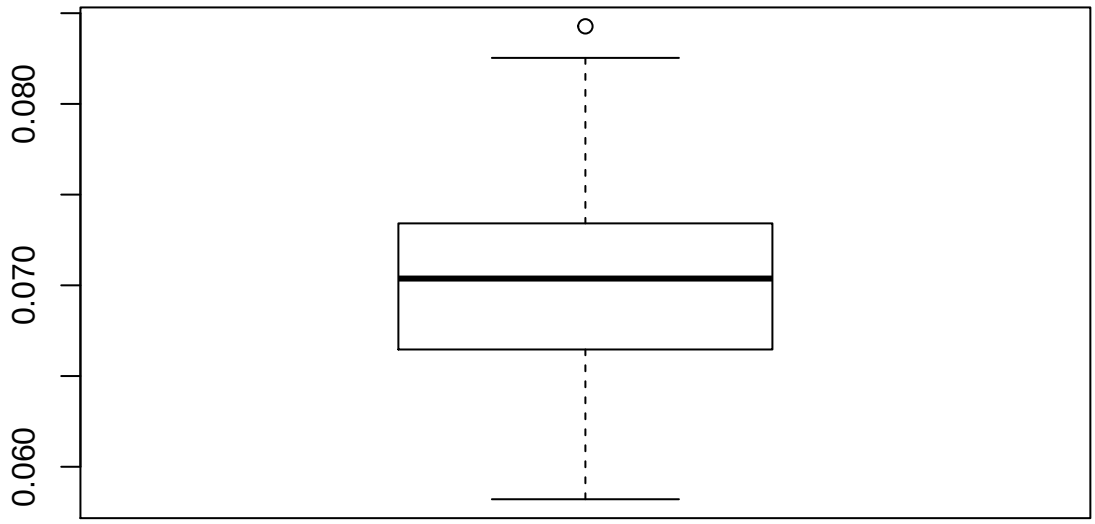
Sur 20 itérations, l'erreur minimum est de 5%, l'erreur maximum est de 8% et l'erreur moyenne de 7%.

Affichons l'histogramme et la boite à moustache des erreurs pour le noyau gaussien.

```r
hist(error, col="firebrick1")
```




```
boxplot(error)
```



En conclusion, un noyau gaussien est mieux adapté.

11. Comparaison des performances des 3 noyaux

Comparons les performances des 3 noyaux gaussien, polynomial et linéaire sur 40 itérations

```
errorLinear= c()
errorPoly = c()
errorGauss = c()

for(k in 1:40){
  errorLinear[k] = getError("vanilladot")
  errorPoly[k] = getError("polydot")
  errorGauss[k] = getError("rbfdot")
}
```

```
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
```

```
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
```

```
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
```

```
## Noyau Lin?aire :
# Erreur minimum :
print(errorLinear[which.min(errorLinear)])
```

```
## [1] 0.05734144
```

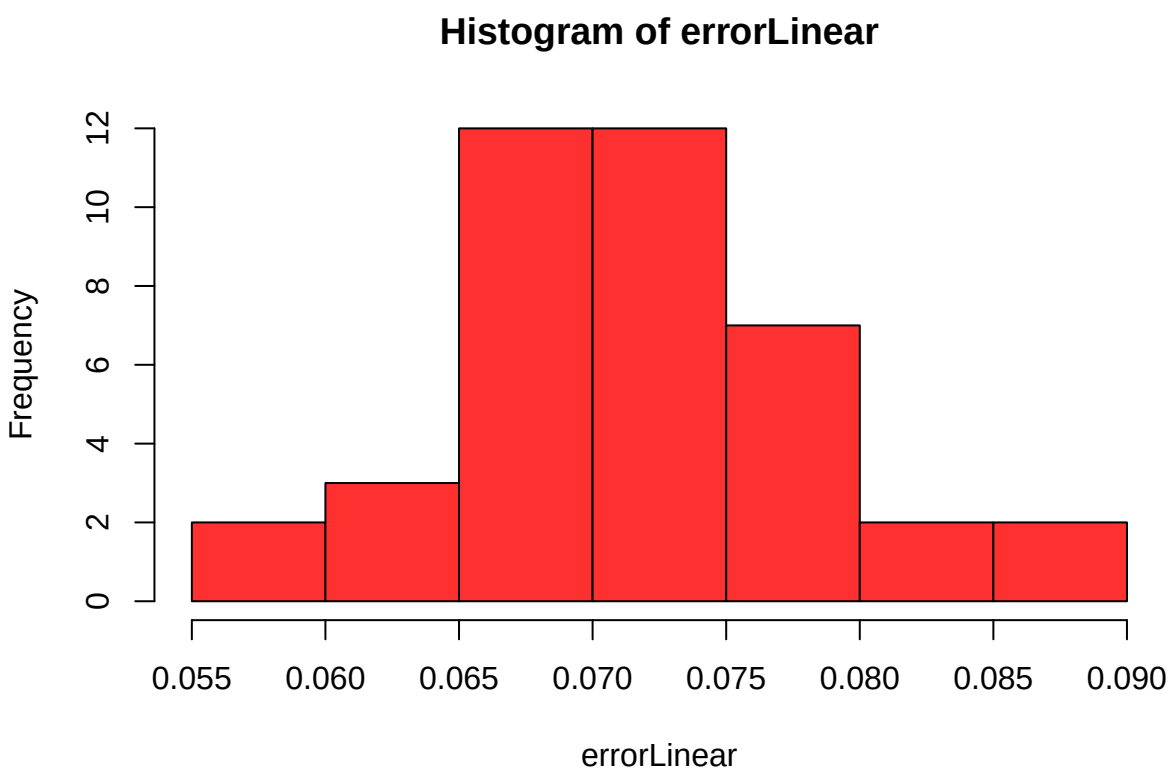
```
# Erreur maximum :
print(errorLinear[which.max(errorLinear)])
```

```
## [1] 0.08774978
```

```
# Erreur moyenne :
print(mean(errorLinear))
```

```
## [1] 0.0717854
```

```
#Histogramme de l'erreur
hist(errorLinear, col="firebrick1")
```




```
## Noyau Polynomiale :
#Erreur minimum :
print(errorPoly[which.min(errorPoly)])
```

```
## [1] 0.05299739
```

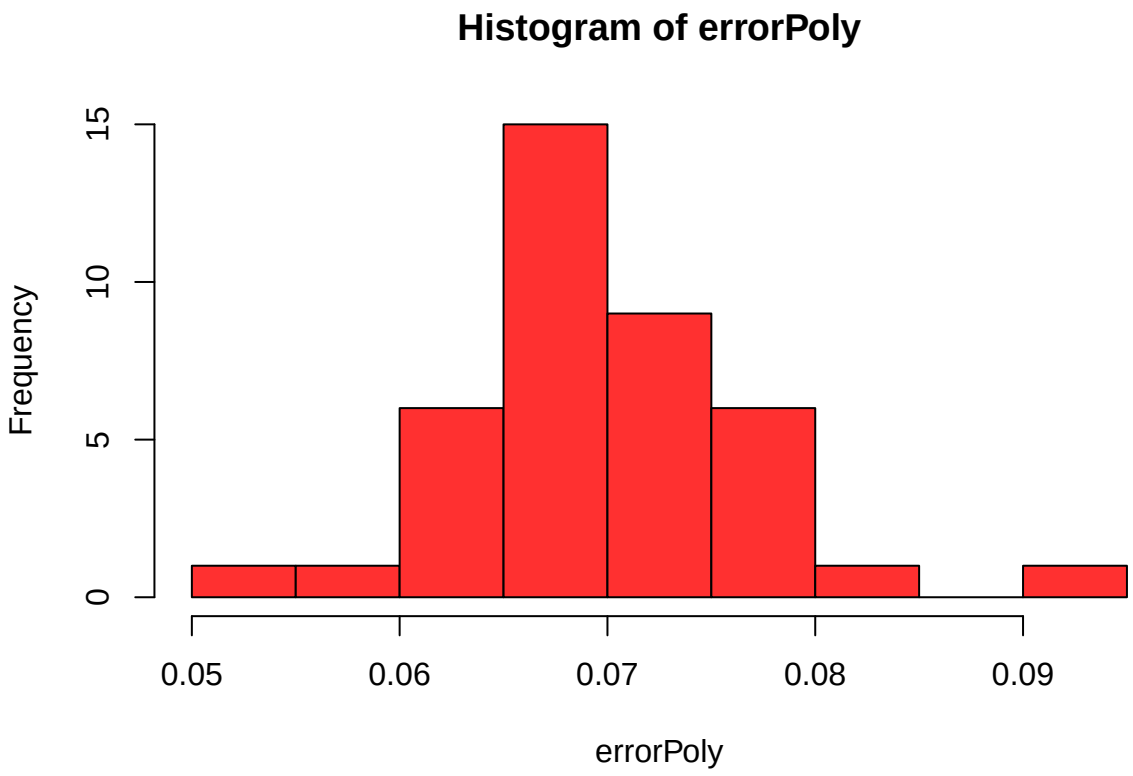
```
# Erreur maximum :
print(errorPoly[which.max(errorPoly)])
```

```
## [1] 0.09035621
```

```
# Erreur moyenne :
print(mean(errorPoly))
```

```
## [1] 0.06939618
```

```
#Histogramme de l'erreur
hist(errorPoly, col="firebrick1")
```




```
## Noyau Gaussien :
#Erreur minimum :
print(errorGauss[which.min(errorGauss)])
```

```
## [1] 0.05734144
```

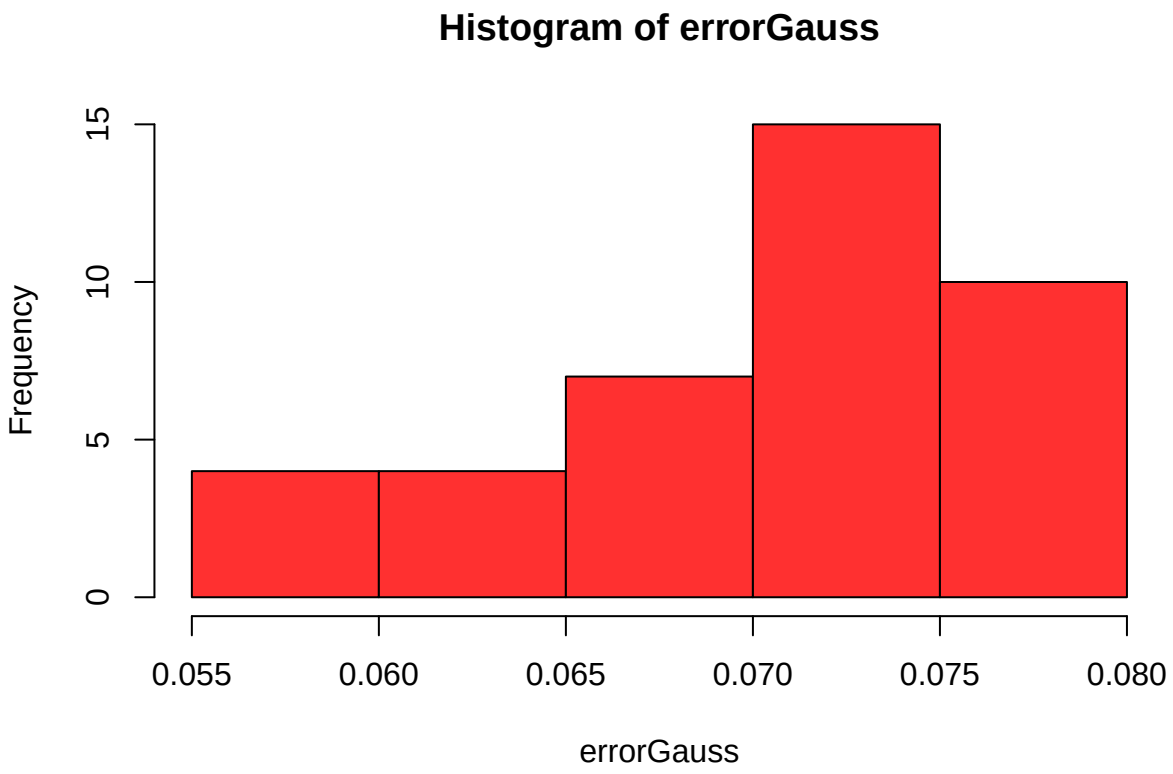
```
# Erreur maximum :
print(errorGauss[which.max(errorGauss)])
```

```
## [1] 0.07906169
```

```
# Erreur moyenne :
print(mean(errorGauss))
```

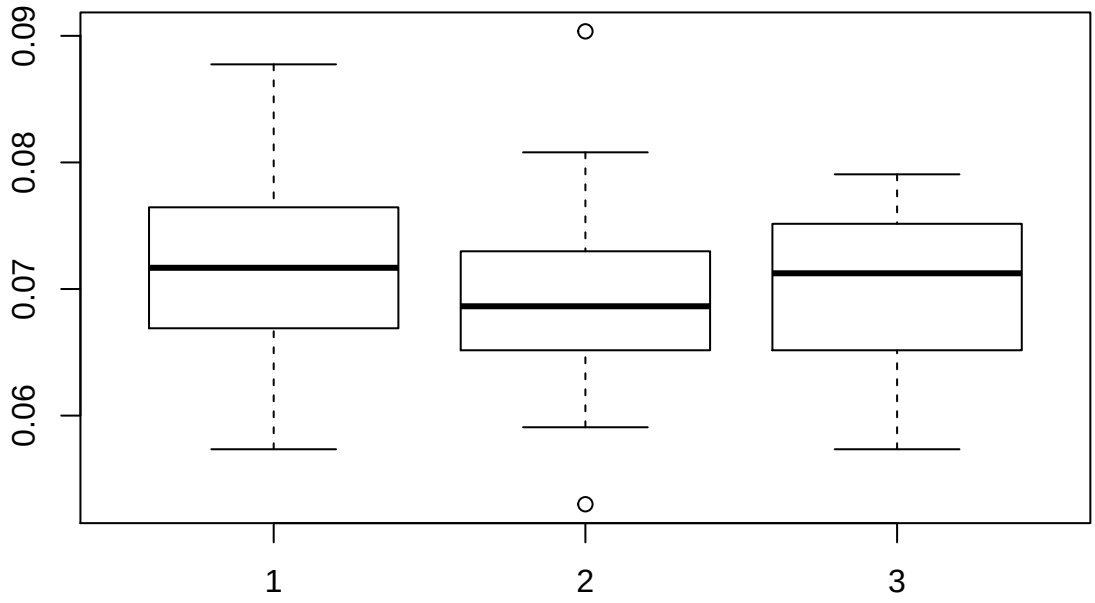
```
## [1] 0.07011295
```

```
#Histogramme de l'erreur
hist(errorGauss, col="firebrick1")
```



Comparons empiriquement les erreurs des 3 noyaux:

```
boxplot(errorLinear, errorPoly, errorGauss)
```



En conclusion le noyau présentant l'erreur la plus basse est le noyau gaussien.

Application III : Modèle de régression par SVM

1. Noyau et performance de UsCrime

Procédons au chargement du fichier *UsCrime.txt* :

```
usCrimeData = read.table('UsCrime.txt',header=TRUE, sep=' ')
#names(usCrimeData) <- NULL # suppression des headers
```

Comparons différents modèles de régression permettant de prédire le taux de crimminalité à l'aide d'un modèle SVM.

```
# library(kernlab)
# help(ksvm)
# getErrorUsCrime = function(NoyauUs = "rbfdot"){
#   # 1) Choix de la base
#   dtusCrime = sort(sample(nrow(usCrimeData), 0.75*nrow(usCrimeData)))
#   trainusCrimeData75<-usCrimeData[dtusCrime,]
#   testusCrimeData25<-usCrimeData[-dtusCrime,]
#
#   # Initialisation des matrices
#   XtrainUsCrime = as.matrix(trainusCrimeData75[,-1])
```

```
#   YtrainUsCrime = as.matrix(trainusCrimeData75[,1])
#   XtestUsCrime = as.matrix(testusCrimeData25[,-1])
#   YtestUsCrime = as.matrix(testusCrimeData25[,1])
#
#   # 2) Calibration du mod?le sur la base d'apprentissage
#   ksvmTrainUsCrime = ksvm(XtrainUsCrime, YtrainUsCrime, kernel=NoyauUs, type="C-svc")
#   #BUG "dependent variable has to be of factor or integer type for classification mode."
#
#   # 3) Evaluation de l'erreur sur la base de test
#   predictTestUsCrime = predict(ksvmTrainUsCrime, XtrainUsCrime)
#   matrixConfUsCrimeTest = table(predictTestUsCrime, YtestUsCrime)
#
#   # Retourne l'erreur
#   return (as.numeric((matrixConfUsCrimeTest[1,2]+matrixConfUsCrimeTest[2,1]) / (matrixConfUsCrimeTest
# }
#
#
# errorusCrimeLinear= c()
# errorusCrimePoly = c()
# errorusCrimeGauss = c()
#
# for(k in 1:1){
#   errorusCrimeLinear[k] = getErrorUsCrime("vanilladot")
#   errorusCrimePoly[k] = getErrorUsCrime("polydot")
#   errorusCrimeGauss[k] = getErrorUsCrime("rbfdot")
# }
#
# ## Noyau Lin?aire :
# # Erreur minimum :
# print(errorusCrimeLinear[which.min(errorusCrimeLinear)])
# # Erreur maximum :
# print(errorusCrimeLinear[which.max(errorusCrimeLinear)])
# # Erreur moyenne :
# print(mean(errorusCrimeLinear))
# #Histogramme de l'erreur
# hist(errorusCrimeLinear, col="firebrick1")
#
# ## Noyau Polynomiale :
# #Erreur minimum :
# print(errorusCrimePoly[which.min(errorusCrimePoly)])
# # Erreur maximum :
# print(errorusCrimePoly[which.max(errorusCrimePoly)])
# # Erreur moyenne :
# print(mean(errorusCrimePoly))
# #Histogramme de l'erreur
# hist(errorusCrimePoly, col="firebrick1")
#
# ## Noyau Gaussien :
# #Erreur minimum :
# print(errorusCrimeGauss[which.min(errorusCrimeGauss)])
# # Erreur maximum :
# print(errorusCrimeGauss[which.max(errorusCrimeGauss)])
# # Erreur moyenne :
```

```
# print(mean(errorusCrimeGauss))
# #Histogramme de l'erreur
# hist(errorusCrimeGauss, col="firebrick1")
```

Comparons empiriquement les erreurs des 3 noyaux:

```
# boxplot(errorusCrimeLinear, errorusCrimePoly, errorusCrimeGauss)
```

En conclusion le noyau gaussien présente l'erreur la plus faible. C'est donc le modèle le plus significatif.