Université de Technologie de Compiègne
SY09 : Analyse des données et Apprentissage automatique

# SY09 : TP 09 : Régression Logistique

Julien Jerphanion

Printemps 2018

## Table des matières

# 1 Programmation

## 1.1 Régression Logistique

On va implémenter ici deux fonctions `log.app` et `log.val` pour respectivement l'apprentissage et le classement du modèle de régression logistique.

### 1.1.1 Apprentissage

On implémente la fonction qui permet de calculer les proba à posteriori.

```
postprob = function(beta, X) {
    X = as.matrix(X)

    sigmoid = function(x) 1 / (1 + exp(-x))

    prob = sigmoid(X %*% beta)
}
```

```
log.app = function(Xapp, zapp, intr, epsi) {
  #' Apprentissag du modèle de régression logistique
  #' @param Xapp jeu de données d'apprentissage
  #' @param zapp étiquettes du jeu de données
  #' @param intr boolean pour l'ajout de paramètres
  #' @param espi précision pour la convergence
  library(MASS)
  # Paramètres
    n = dim(Xapp)[1]
    p = dim(Xapp)[2]

    Xapp = as.matrix(Xapp)

    if (intr == TRUE) {
        Xapp = cbind(rep(1,n),Xapp)
        p = p + 1
    }

    # Changement d'étiquette (2,1) → (0,1)
    targ = matrix(as.numeric(zapp),nrow=n)
    targ[which(targ==2),] = 0
    tXap = t(Xapp)

    # Paramètres à estimer
    beta = matrix(0,nrow=p,ncol=1)

    conv = FALSE
    iter = 0
    while (conv == FALSE) {
        iter = iter + 1
        bold = beta

        prob = as.numeric(postprob(beta, Xapp))
        MatW = diag((1-prob)*prob)
```

```
        beta = beta + ginv(tXap %*% MatW %*% Xapp) %*% tXap %*% (targ - prob)

        if (norm(beta-bold)<epsi) {
            conv = TRUE
        }
    }

    prob = postprob(beta, Xapp)
    out = NULL
    out$beta = beta
    out$iter = iter
    out$logL = sum(targ * log(prob) + (1-targ) * log(1-prob))

    out
}
```

### 1.1.2 Classement

```
log.val = function(beta, Xtst) {
  #' Classement pour le modèle de régression logistique
  #' @param beta paramètres du modèle de régression
  #' @param Xtst jeu de données à classer
    m = dim(Xtst)[1]
    p = dim(beta)[1]
    pX = dim(Xtst)[2]

    Xtst = as.matrix(Xtst)

    if (pX == (p-1)) {
        Xtst  = cbind(rep(1,m),Xtst)
    }

    prob1 = postprob(beta, Xtst)

    prob = cbind(prob1,(1-prob1))
    pred = max.col(prob)

    out = NULL
    out$prob = prob
    out$pred = pred

    return(out)
}
```

### 1.1.3 Régresion logistique quadratique

```
quadratize = function(X) {
  #' Quadratise un jeu de données 2D
  #' @param X jeu de données 2D à quadratiser

  Y = X
```

```
  for (p in 1:(dim(X)[2]-1)) {
    for (q in (p+1):dim(X)[2]) {
      Y = cbind(Y, X[,p]*X[,q])
    }
  }

  for (p in 1:dim(X)[2]) {
      Y = cbind(Y, X[,p]^2)
  }

  Y
}
```

```
log.quad.app = function(Xapp, zapp, intr, epsi) {
  #' Quadratize les données et procède à l'apprentissage
  #' @param Xapp jeu de données d'apprentissage à quadratiser
  #' @param zapp étiquettes du jeu de données
  #' @param intr boolean pour l'ajout de paramètres
  #' @param espi précision pour la convergence
  Yapp = quadratize(Xapp)

  log.app(Yapp,zapp,intr,epsi)
}
```

```
log.quad.val = function(beta,Xtst) {
  #' Quadratize les données et procède au classement
  #' @param beta paramètres du modèle de régression
  #' @param Xtst jeu de données à classer à quadratiser
  Ytst = quadratize(Xtst)

  log.val(beta,Ytst)
}
```

## 1.2 Vérification des fonctions

```
source("./fonctions/prob.log.R")
source("./fonctions/prob.log2.R")
```

On effectue la regression sur le jeu de données test.

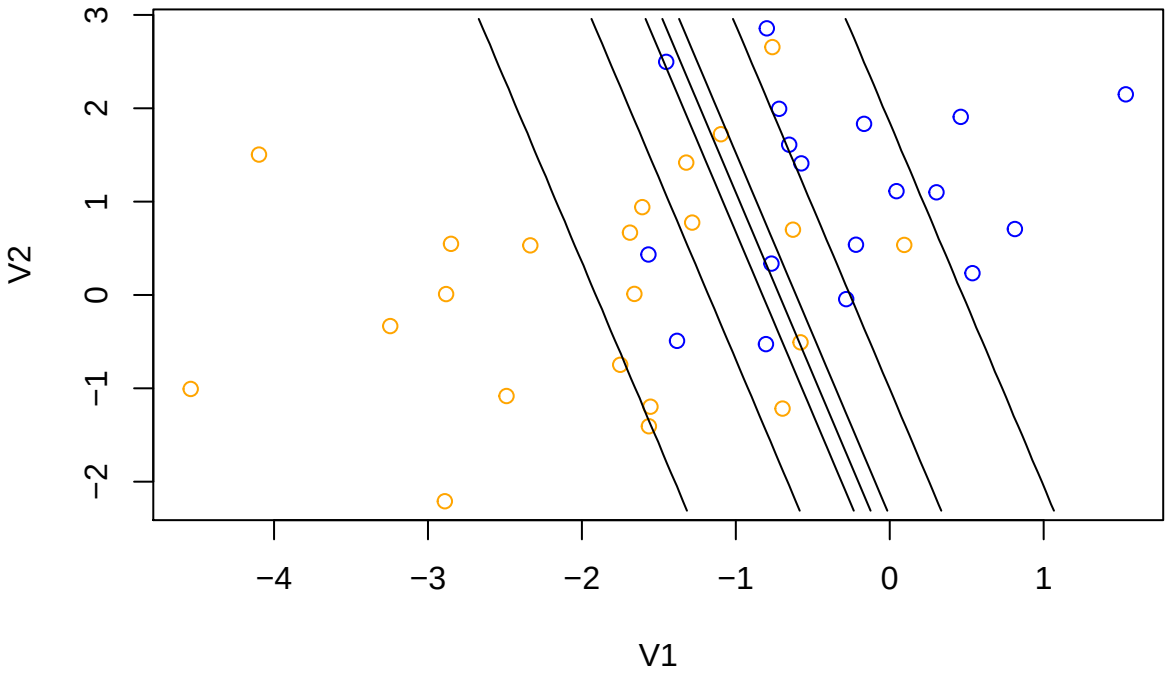
```
data = read.csv("../TP06/donnees/Synth1-40.csv")
X = data[,1:2]
Z = data[,3]

params.log = log.app(X,Z,TRUE,10e-5)
res.log = log.val(params.log$beta, X)

mean(res.log$pred == Z)
```

```
## [1] 0.775
```

```
niveaux = c(0.1, 0.3, 0.45, 0.5, 0.55, 0.7, 0.9)
prob.log(params.log$beta,X,Z,niveaux)
```



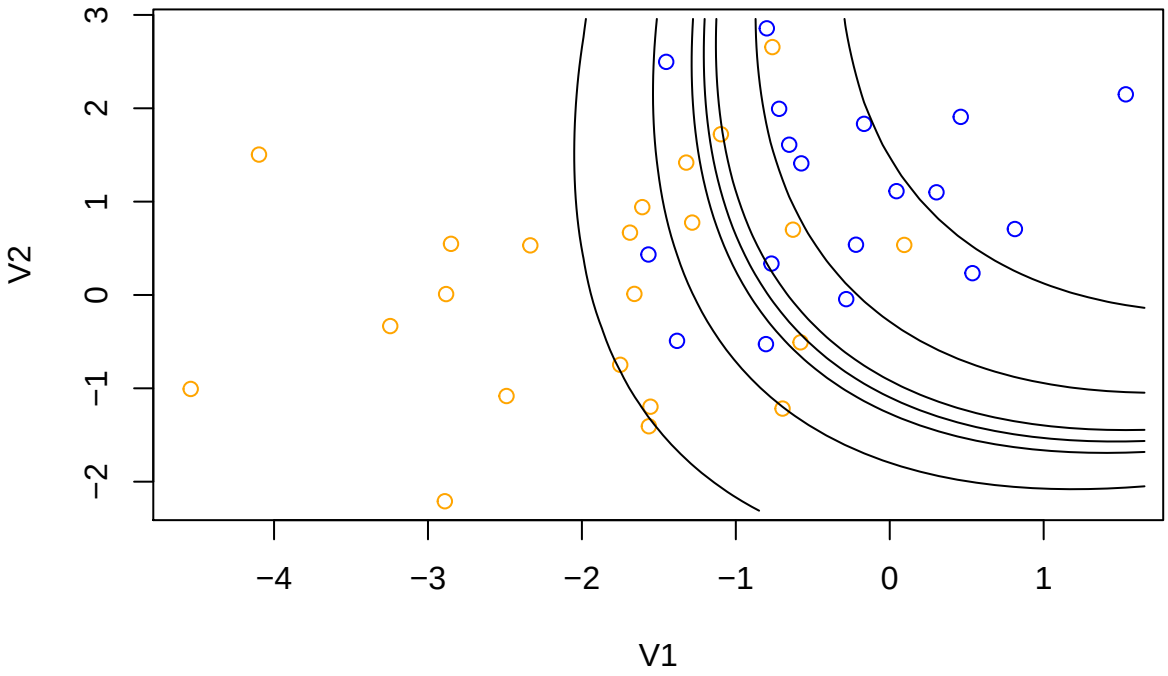

Et sur le jeu de données quadratisé.

```
params.log.quad = log.quad.app(X,Z,TRUE,10e-5)
res.log.quad = log.quad.val(params.log.quad$beta,X)

mean(res.log.quad$pred == Z)
```

```
## [1] 0.8
```

```
niveaux = c(0.1, 0.3, 0.45, 0.5, 0.55, 0.7, 0.9)
prob.log2(params.log.quad$beta,X,Z,niveaux)
```




# 2 Application

## 2.1 Test sur données simulées

On réutilise la fonction de séparation utilisée précédemment.

```
# Pour la création de jeu de données d'apprentissage et de test
source("../TP06/fonctions/separ1.R")
```

On met en place cette fonction pour l'évaluation d'un modèle.

```
log.eval = function(X,z, niter=20, test_size = 1/3,intr=TRUE,epsi=10e-5,model.app=log.app, model.val=log
  #' Détermination des erreurs pour la regression logistique
  #'
  #' @param X : jeu de données
  #' @param z : les étiquettes du jeu de données
  #' @param niter : le nombre d'estimation à réaliser
  #' @param test_size : proportion à considérer pour l'ensemble de test
  #' @param model.app : version de l'apprentissage
  #' @param model.val : version du classement apprentissage

  error = function(z1,z2) {
    #' Calcule l'erreur de classification entre deux
    #' vecteurs d'étiquettes.
    misClassified = 1 * (z1 != z2)
```

```
    sum(misClassified) / length(z1)
  }

  errApp = c()
  errTest = c()
  for (i in 1:niter) {
    donn = separ1(X,z,test_size)
    Xapp = donn$Xapp
    Xtst = donn$Xtst
    zapp = donn$zapp
    ztst = donn$ztst

    params = model.app(Xapp,zapp,intr,epsi)

    zappClass = model.val(params$beta,Xapp)$pred
    ztstClass = model.val(params$beta,Xtst)$pred

    errApp[i] = error(zapp,zappClass)
    errTest[i] = error(ztst,ztstClass)
  }

  estErrApp = mean(errApp)
  estErrTest = mean(errTest)

  # Objet de retour
  errors = NULL
  errors$estErrApp = estErrApp
  errors$estErrTest = estErrTest

  errors$errApp = errApp
  errors$errTest = errTest

  errors
}
```

On effectue les évaluations sur les jeux de données

```
datasets = c("Synth1-1000.csv",
             "Synth2-1000.csv",
             "Synth3-1000.csv")

ms = c()
errApp = c()
errTst = c()
errApp.quad = c()
errTst.quad = c()
count = 1
for (d in datasets) {
  print(d)
  donn = read.csv(paste("./donnees/",d,sep = ""))
  p = dim(donn)[2] -1
  X = donn[,1:p]
  Z = donn[,p+1]
```

```
  # Cas normal
  res.eval = log.eval(X,Z,model.app = log.app, model.val = log.val)
  errApp[count] = res.eval$estErrApp
  errTst[count] = res.eval$estErrTest

  # Cas quadratique
  res.eval.quad = log.eval(X,Z,model.app = log.quad.app, model.val = log.quad.val)

  errApp.quad[count] = res.eval.quad$estErrApp
  errTst.quad[count] = res.eval.quad$estErrTest

  count = count + 1
}
```

```
## [1] "Synth1-1000.csv"
## [1] "Synth2-1000.csv"
## [1] "Synth3-1000.csv"
```

```
df.res = data.frame(DataSet = datasets,
                    Err.App.Linr = errApp,
                    Err.Tst.Linr = errTst,
                    Err.App.Quad = errApp.quad,
                    Err.Tst.Quad = errTst.quad)

df.res
```

```
##           DataSet Err.App.Linr Err.Tst.Linr Err.App.Quad Err.Tst.Quad
## 1 Synth1-1000.csv   0.03521021   0.03113772   0.03145646   0.03458084
## 2 Synth2-1000.csv   0.06964018   0.07147147   0.06034483   0.06351351
## 3 Synth3-1000.csv   0.08545727   0.08633634   0.08320840   0.08813814
```

## 2.2 Test sur données réelles

### 2.2.1 Données 'Pima

```
donn = read.csv("./donnees/Pima.csv", header=T)
X = donn[,1:7]
Z = donn[,8]

# Cas normal
res.eval = log.eval(X,Z,model.app = log.app, model.val = log.val,niter = 100)
res.eval$estErrApp
```

```
## [1] 0.2074366
```

```
res.eval$estErrTest
```

```
## [1] 0.2219774
```

```
# Cas quadratique
res.eval.quad = log.eval(X,Z,model.app = log.quad.app, model.val = log.quad.val, niter = 100)

res.eval.quad$estErrApp
```

```
## [1] 0.1922254
```

```
res.eval.quad$estErrTest
```

```
## [1] 0.2398305
```

### 2.2.2 Données `breast cancer Wiscoinsin`

```
donn = read.csv("./donnees/bcw.csv", header=T)
X = donn[,1:9]
Z = donn[,10]

# Cas normal
res.eval = log.eval(X,Z,model.app = log.app, model.val = log.val,niter = 100)
res.eval$estErrApp
```

```
## [1] 0.0323956
```

```
res.eval$estErrTest
```

```
## [1] 0.04074561
```

```
# Cas quadratique
```

.