TP5

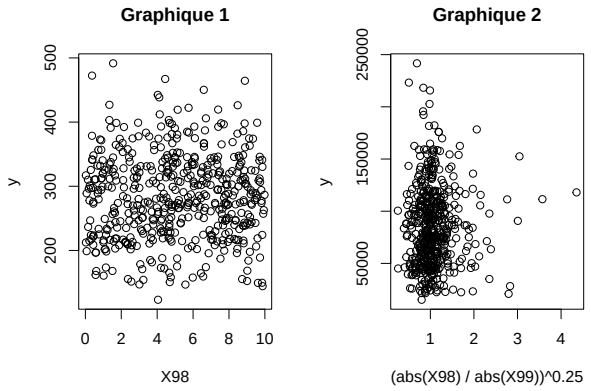
Raphael Nguyen - Tristan Hucher

Problème de Régression

```
donnees <- read.table("TP5_a23_reg_app.txt")
```

Importation et visualisation des données On remarque qu'on ne peut dire grand chose en affichant juste la variable à expliquer en fonction des variables (graphique 1). On a donc manipuler les variables dans tous les sens pour faire ressortir des liens pertinents avec y. Par exemple : en prenant la racine carré de la racine carré du ratio entre les variables 2 à 2, on isole un peu certaines données (graphique 2).

```
#Exploration des données :
#boxplot(donnees[,-101])
for (i in 1:99){
  #plot(donnees[,i],donnees[,101])
  #plot((abs(donnees[,i])/abs(donnees[,i+1]))^0.25,donnees[,101]^2)
}

par(mfrow = c(1,2))
# Premier graphique
plot(donnees[, 98], donnees[, 101], main = "Graphique 1", xlab = "X98", ylab = "y")

# Deuxième graphique
plot((abs(donnees[, 98]) / abs(donnees[, 99]))^0.25, donnees[, 101]^2, main = "Graphique 2", xlab = "(ab
```



Création d'échantilles d'apprentissage et de test

```
n <- nrow(donnees)
p <- ncol(donnees)

n.train <- floor(4/5 * n)
index.train <- sample(n,size = n.train , replace = FALSE)
```

```
donnees.train <- donnees[index.train, ]
donnees.test <- donnees[-index.train, ]

k <-10
fold <- sample(k,n,replace = TRUE)
```

On va ajouter de nouvelles variables qui pourront permettre de mieux expliquer y puis on va les selectionner avec le critère BIC pour avoir que celles qui sont pertinentes. Nous avons également tester d'ajouter des splines naturelles mais ca n'apportait rien en terme de MSE.

```
donnees2 <- donnees[,-101]
n_colonnes <- ncol(donnees)

# Ajouter les colonnes au carré
for (i in 1:n_colonnes-1) {
  nom_colonne <- colnames(donnees2)[i]
  nom_colonne2 <- colnames(donnees2)[i+1]
  donnees2[paste0(nom_colonne, "/",nom_colonne2)] <- (abs(donnees2[,i])/abs(donnees2[,i+1]))^0.25
}
donnees2 <- cbind(donnees2, y = donnees[,"y"])

model_lm = lm(y~.-y, data = donnees2)
model_BIC <- stepAIC(model_lm,scope=y~.-y,direction="both",k=log(nrow(donnees2)))

#model_BIC <- stepAIC(model_lm,scope=y~.+ns(X2,5)+ns(X4,5)+ns(X6,5)+ns(X8,5)+ns(X10,5)+ns(X12,5)+ns(X13
```

On a sélectionné des variables à l'aide de stepBIC et obtenu une sélection de variables pertinentes. (par soucis d'économie d'espace on ne précisera pas le code du BIC). On conserve les variables suivantes :

```
donnees3<-donnees2[,c("X4","X6","X8","X10","X12","X13","X14","X16","X18","X20","X21","X22","X24","X26",
```

On va tester plusieurs modèles et à chaque fois on testera des variantes. On testera : - juste les variables de base - les variables augmentées puis selectionnées par BIC - normalisation des données en entrée.

```
#Linear model , Ridge , Lasso , Elastic Net
MSE_LM<- rep(0,k)
MSE_Ridge <- rep(0,k)
MSE_Lasso <- rep(0,k)
MSE_Ennet <- rep(0,k)
donnees2<-donnees3
for (i in 1:k){

  #traitement

  #sans normalisation
  #X.app <- data.frame(donnees2[fold !=i,])
  #X.tst <- data.frame(donnees2[fold ==i,])

  #avec normalisation

  sd_cols <- colSds(as.matrix(donnees2[fold !=i,-68]))
  mean_cols <-colMeans(donnees2[fold !=i,-68])
```

```r
  X.app <- as.data.frame(t(t(sweep(donnees2[fold !=i,-68], 2, mean_cols, `-`))/sd_cols))
  X.tst <- as.data.frame(t(t(sweep(donnees2[fold ==i,-68], 2, mean_cols, `-`))/sd_cols))

  X.app <- cbind(X.app, y = donnees2[fold !=i,"y"]) #ajout du Y qu'on avait enlevé
  X.tst <- cbind(X.tst, y = donnees2[fold ==i,"y"])

  x <- model.matrix(y~.,X.app)
  x_test <- model.matrix(y~.,X.tst)
  y_train <- X.app[,"y"]

  #Linear model
  modellm <- lm(y~., data = X.app)
  pred_lm<-predict(modellm,newdata=X.tst)
  MSE_LM[i] <- mean((X.tst$y-pred_lm)^2)

  #ridge
  cv.out<-cv.glmnet(x,y_train,alpha=0)

  model_ridge<-glmnet(x,y_train,lambda=cv.out$lambda.min,alpha=0)
  pred_rigde<-predict(model_ridge,s=cv.out$lambda.min,newx=x_test)
  MSE_Ridge[i] <- mean(((X.tst$y-pred_rigde)^2))

  #Lasso
  cv.out_lasso<-cv.glmnet(x,y_train,alpha=1)

  model_lasso<-glmnet(x,y_train,lambda=cv.out_lasso$lambda.min,alpha=1)
  pred_lasso<-predict(model_lasso,s=cv.out_lasso$lambda.min,newx=x_test)
  MSE_Lasso[i] <- mean((X.tst$y-pred_lasso)^2)

  #elastic-net

  train_control <- trainControl(method = "repeatedcv",
                                number = 5,
                                repeats = 5,
                                search = "random",
                                verboseIter = FALSE)

  elastic_net_model <- train(y~.,
                             data = X.app,
                             method = "glmnet",
                             tuneLength = 25,
                             trControl = train_control)

  pred_enet <- predict(elastic_net_model, newdata = X.tst)
  MSE_Ennet[i] <- mean(((X.tst$y-pred_enet)^2))
}
mean(MSE_LM)
mean(MSE_Ridge)
mean(MSE_Lasso)
mean(MSE_Ennet)

#résultats :
```

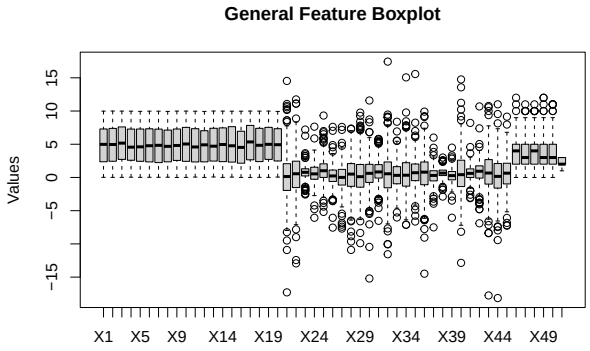
```
#Variables Non normalisées :
#LM : 200.1338 ; Ridge : 205.7189 ; Lasso : 190.4084 ; Elastic-net : 192.0821

#Variables Non normalisées :
#LM : 200.1338 ; Ridge : 205.7189 ; Lasso : 190.3206 ; Elastic-net : 192.582

#variables selectionnées et non Normalisées
#LM : 114.0967 ; Ridge : 120.3131 ; Lasso : 113.991 ; Elastic-net : 113.9955
```

On en conclut que le Lasso est plus performant que les autres méthodes, c'est celle ci que nous allons utiliser.

Problème de Classification

```
data = read.table("TP5_a23_clas_app.txt")
data$y = as.factor(data$y) # indique clairement qu'il s'agit de classes
```

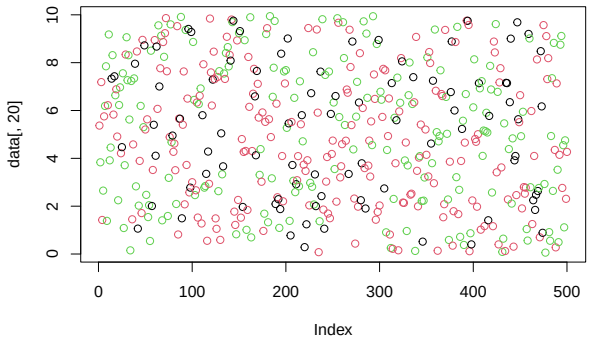
```
# summary(data)

boxplot(data, main = "General Feature Boxplot", ylab = "Values")
```



Importation et visualisation des données

```
# boxplot(data[data$y==1,], main = "Y = 1 Feature Boxplot", ylab = "Values")
# boxplot(data[data$y==2,], main = "Y = 2 Feature Boxplot", ylab = "Values")
# boxplot(data[data$y==3,], main = "Y = 3 Feature Boxplot", ylab = "Values")
```

```
for (i in 1:(length(data)-1)) {
  # plot(data[, i], col = data$y)
}
plot(data[, 20], col = data$y)
```



Domaines des variables :

- Les variables X1 à X20 prennent des valeurs réelles strictement positives
- X21 à X45 sont réelles positives ou négatives
- Les variables X46 à X50 sont des entiers positifs

Structure :

- La dispersion des données est très similaire au sein de chacune des trois familles de classes.
- Aucune variable n'a de donnée absente ou visiblement aberrante.
- Il y a des différences visibles entre chaque classe sur la plupart des variables, les points de données vertes (3e classe) semblent notamment souvant davantage dispersés.
- Les points de données noirs (1ère classe) ont l'air d'être sous représentés. (confirmé ci-dessous)

```
total = length(data[,1])
for (i in 1:3) {
  class_len = length(data[data$y == i,1])
  cat("effectif classe ",i," : ",class_len,"(",round(class_len/total,2),")\n")}
```

```
## effectif classe  1  :  90 ( 0.18 )
## effectif classe  2  :  220 ( 0.44 )
## effectif classe  3  :  190 ( 0.38 )
```

Construction des ensembles Train / Test

```
K = 5 # number of folds to be used
n = dim(data)[1]
folds = sample(K, n, replace = TRUE)
```

Evaluation de différentes méthodes sans normalisation ni sélection de variable

```r
evaluate_fold = function(train, test, K, errors) { # performs analysis for each technique and each fold
  p = ncol(data) - 1

  ytest = test$y

  fit = multinom(formula = y ~ ., data = train, family = binomial)
  yhat = predict(fit, newdata = test)
  errors[1] = errors[1] + mean(yhat != ytest)/K # Multinomial Log Regression

  fit = qda(formula = y ~ ., data = train)
  yhat = predict(fit, newdata = test)$class
  errors[2] = errors[2] + mean(yhat != ytest)/K # QDA

  fit = lda(formula = y ~ ., data = train)
  yhat = predict(fit, newdata = test)$class
  errors[3] = errors[3] + mean(yhat != ytest)/K # LDA

  fit = naive_bayes(y~., data = train)
  yhat = predict(fit, newdata = test)
  errors[4] = errors[4] + mean(yhat != ytest)/K # Naive Bayes

  fit = randomForest(as.factor(y) ~ . , data = train, mtry = p / 2, importance = TRUE)
  yhat = predict(fit, newdata=test, type="response")
  errors[5] = errors[5] + mean(yhat != ytest)/K # Random Forest

  return(errors)
}

evaluate = function(data, K, errors) { # general evaluation function for further use
  for (k in 1:K) {
    train = data[folds != k,]
    test = data[folds == k,]
    errors = evaluate_fold(train, test, K, errors)
  }
  return(errors)
}
```

```r
methode_names = c("Regression Multinomiale", "QDA", "LDA", "Naive Bayes", "Random Forest")
N = length(methode_names) # number of different techniques used
K = 5 # number of folds to be used
errors = rep(0, N) # each row represents the results by fold for a particular technique.

errors = evaluate(data, K, errors) # running the training and predict process
```

```r
for (n in 1:N) { # displaying the results
  cat(methode_names[n], ":", round(errors[n], 2), "\n")}
```

```
## Regression Multinomiale : 0.43
## QDA : 0.35
## LDA : 0.43
## Naive Bayes : 0.35
## Random Forest : 0.4
```

Réevaluation après sélection de variable par le critère AIC

On souhaite en particulier améliorer les performances de certaines techniques, notamment les analyses discriminantes et la log régression multinomiale.

On préfère le critère AIC car le nombre de variables n'est pas très élevé, on souhaite en conserver plus qu'avec BIC qui est plus pénalisant.

```
# fit = multinom(formula = y ~ ., data = data, family = "binomial") # the AIC is defined for this linea
# AIC = stepAIC(fit, scope=y~.-y, direction="both")
AICselected = c("X4", "X6", "X13", "X24", "X27", "X29", "X30", "X32", "X33", "X46", "X47", "X48", "X49"
AICdata = data[,AICselected] # creating a dataset with Only the selected features
```

```
errors = rep(0, N)
errors = evaluate(AICdata, K, errors) # running the training and predict process with the new dataset
```

```
for (n in 1:N) { # displaying the results
  cat(methode_names[n], ":", round(errors[n], 2), "\n")}
```

```
## Regression Multinomiale : 0.39
## QDA : 0.39
## LDA : 0.37
## Naive Bayes : 0.37
## Random Forest : 0.4
```

Seuls la regression multinomiale et la LDA ont de meilleures performances. La QDA est négativement affectée.

Réevaluation après normalisation des données

Nous allons désormais normaliser les données au préalable :

```
evaluate_normalized = function(data, K, errors) { # performs normalization of the data before train/tes
  for (k in 1:K) {
    train = scale(data[folds != k,-51])
    test = scale(data[folds == k,-51])

    train = as.data.frame.matrix( cbind(train, y = data[folds != k, 51]) ) # putting back y
    test = as.data.frame.matrix( cbind(test, y = data[folds == k, 51]) )

    train$y = as.factor(train$y)
    test$y = as.factor(test$y)

    errors = evaluate_fold(train, test, K, errors)
  }
  return(errors)
}
errors = rep(0, N)
errors = evaluate_normalized(data, K, errors) # running the training and predict process with the new d
```

```r
for (n in 1:N) { # displaying the results
  cat(methode_names[n], ":", round(errors[n], 2), "\n")}
```

```
## Regression Multinomiale : 0.42
## QDA : 0.35
## LDA : 0.42
## Naive Bayes : 0.37
## Random Forest : 0.41
```

La normalisation ne semble pas avoir d'effet particulier.

On choisit donc de conserver une QDA sans sélection de variable et sans normalisation.