SY19 - TP note - Regression et classification

Jingyi HU

November 13, 2017

Classification

Dans cette première partie, nous allons décrire notre méthologie pour construire notre modèle de classfication. Vous pouvez trouver tous les graphiques associés dans l'archivre nommé pdf.(De plus, si vous voulez vous pouvez entrer dans le fichier .RMD pour exécuter les scripts.Changement de 'working path' : setwd("votre path"))

Tout d'abord, nous avons fait importation et des découverts des données, et nous voyons bien que les y sont des "1" ou "2". Donc, il s'agit un problème de classification.

```
X <- read.table('tp3_clas_app.txt', header = TRUE)
X$y
```

```
##   [1] 1 2 1 2 2 2 1 2 1 2 1 2 2 1 2 2 1 2 1 2 1 2 2 2 2 2 2 2 2 2 2 1 1 2 2
##  [36] 1 1 2 2 2 2 2 2 2 2 2 2 1 2 1 2 1 1 2 2 1 2 1 2 2 2 1 2 1 1 1 1 2 2 1
##  [71] 2 2 2 1 2 2 2 1 2 2 2 1 2 2 2 1 2 2 2 1 1 1 2 2 1 2 1 2 2 1 1 2 2 2 2
## [106] 2 1 2 2 2 1 1 1 1 2 1 1 1 2 2 2 1 2 1 1 2 2 1 1 1 2 1 2 1 2 2 1 2 1 2
## [141] 2 2 2 2 2 2 2 1 2 2 1 1 1 2 1 1 2 1 2 2 2 2 1 2 1 2 2 2 1 1 2 2 1 2 1
## [176] 2 1 2 1 2 2 1 2 2 1 2 1 2 2 2 1 1 1 1 2 2 2 2 1 2
```

```
head(X, 2)
```

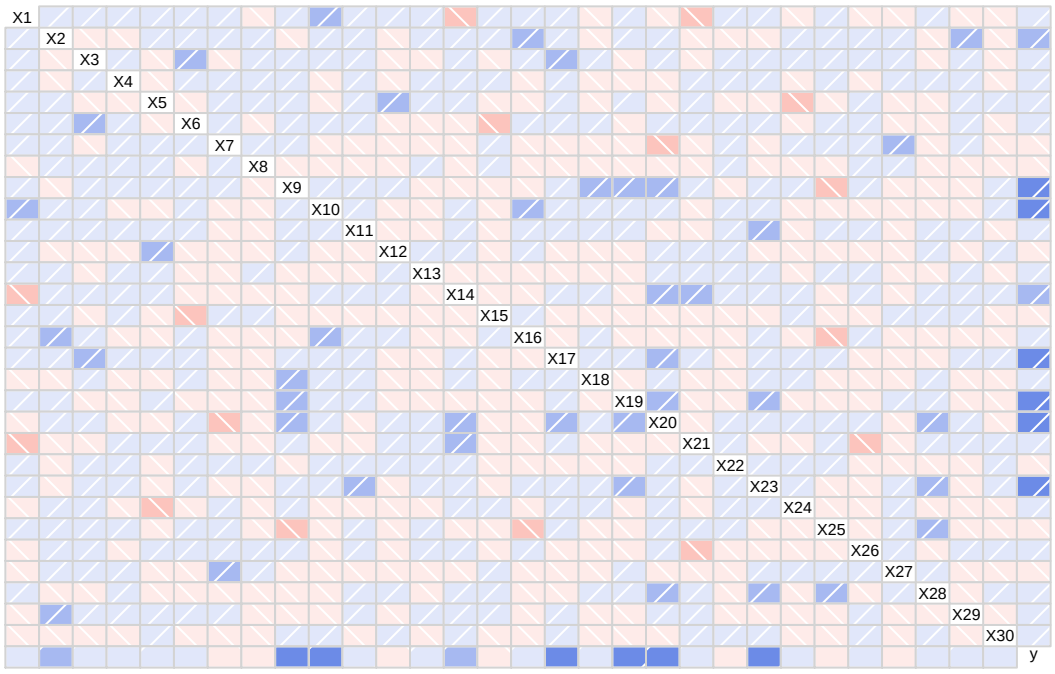
```
##           X1         X2         X3         X4        X5         X6
## 1 -0.1635431 -0.4175766  0.8017694  1.8257193 0.5086949 -0.7883203
## 2 -1.4348931  0.8250970 -0.2347362 -0.7189092 0.4362584 -0.5797873
##           X7         X8        X9       X10        X11        X12
## 1  0.4680607  1.0811583 -0.584323 -1.282427  0.2430432 -0.6638328
## 2 -0.1287590 -0.2645659  0.380593  0.186444 -0.1904827  0.7945773
##         X13         X14        X15         X16        X17        X18
## 1 0.6633116  0.10608738  0.2722818  0.01060115 0.56717224 -0.1258052
## 2 0.3163512 -0.08258698 -0.6833113 -0.79170696 0.05986593 -1.8168842
##          X19      X20        X21       X22       X23          X24
## 1 -0.3195833 0.403135 -0.7507821 1.3585466 0.4750515  0.005758411
## 2  0.5033665 1.127842  0.8121465 0.3252198 1.1057115 -0.537007091
##          X25       X26         X27       X28      X29        X30 y
## 1 -2.1604370 0.6270944 -0.68548917 0.4845073 1.569288 -1.6503772 1
## 2  0.4987767 0.9855485  0.03220178 1.0033308 2.261504 -0.4467132 2
```

D'après le résultat, nous trouvons que toutes les résultats sont quantitatives et il y a 2 classes en présence.(y = 1 ou y = 2).
Nous allons passer à étudier est-ce qu"il existe des **correlations** fortes entre les variables.

```
library(corrgram)
```

```
## Warning: package 'corrgram' was built under R version 3.3.3
```

```
corrgram(X)
```

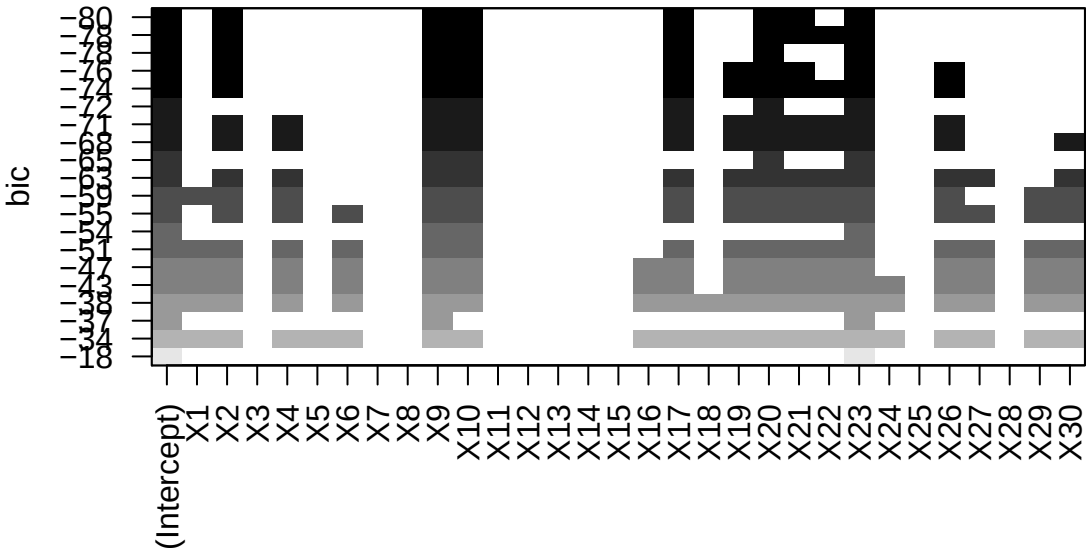


D'après le graphique, nous trouvons que les variables sont corrélées 2 à 2 pas fortement. Donc, nous allons étudier si elles apportent des informations significatives sur y. Autrement dit pouvons nous selectionner un sous-ensemble de variables pour construire notre modèle?(subsets). Normalement, la fonction regsubset est utilisée pour la regression, mais ici nous pouvons l'utiliser pour la classification, car il s'agit d'un problème de classification à deux classes, on peut donc minimiser la somme des carres des erreurs MSE.

```
library('leaps')
```

```
## Warning: package 'leaps' was built under R version 3.3.3
```

```
plot(regsubsets(y ~ ., nvmax = 20, data=X))
```



D'après le graphique, nous pouvons trouver que les variables X2, X9, X10, X17, X20 et X23 soient suffisantes pour construire le modèle.
L'impact d'un subset selection

Methologie générale d'apprentissage d'un modèle

Pour chaque modèle, afin de construire un estimateur sans biais de l'erreur sur l'echantillon, nous utilisons la méthode de l'ensemble de validation qui consiste à partitionner aléatoirement l'échantion en un ensemble d'apprentissage et un ensemble de test, avec une proportion de 2/3, 1/3. Notre fonction est comme ci-dessous:

```
split_dataset <- function(dataset, percentage_training_set) {
  training_rows <- sample(1:nrow(dataset),
                          percentage_training_set * nrow(dataset))

  res <- NULL
  res$train_set <- dataset[training_rows, ]
  res$test_set <- dataset[-training_rows, ]

  res
}
```

Premier méthode: Analysés discriminantes quadratique et lineaire(LDA et QDA)

Supposons qu'il suit conditionnellement une loi normale multidimensionnelle à chaque classe, nous pouvons utiliser LDA et QDA.
Quand est-il pour notre échantillon ?

```r
library(mvShapiroTest)
```

```
## Warning: package 'mvShapiroTest' was built under R version 3.3.2
```

```r
# nous séparons des individus selon leurs classe 1 ou 2(y = 1 ou 2)
X_class1 = as.matrix(X[X$y == 1, 1:30])
X_class2 = as.matrix(X[X$y == 2, 1:30])
# et puis nous faisons un test de Shapiro (test de loi normal multidimensionnelle)
mvShapiro.Test(X_class1)
```

```
##
##  Generalized Shapiro-Wilk test for Multivariate Normality by
##  Villasenor-Alva and Gonzalez-Estrada
##
## data:  X_class1
## MVW = 0.98249, p-value = 0.2768
```

```r
mvShapiro.Test(X_class2)
```

```
##
##  Generalized Shapiro-Wilk test for Multivariate Normality by
##  Villasenor-Alva and Gonzalez-Estrada
##
## data:  X_class2
## MVW = 0.98731, p-value = 0.07145
```

D'après le résultat, nous ne rejettons pas l'hypothese qu'il suit une loi normale multidimensionnelles conditionnellement à la chaque classe.
Sans hypothese suplementaire sur les distributions conditionnelles nous pouvons appliquer l'analyse discriminante lineaire.(LDA)

```r
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.3.3
```

```
## Loading required package: lattice
```

```
## Warning: package 'lattice' was built under R version 3.3.3
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 3.3.3
```

```r
library(MASS)
attach(X)
```

```r
set.seed(1) #pour obtenir le même résultat
folds <- createFolds(y, k = 10, list = TRUE, returnTrain = FALSE)

error_rates_lda <- matrix(nrow = 10, ncol = 1)
#car on a 10 folds

#using 10-folds cross-validation
for (i in 1:10) {
  #make the folds
  train_df_lda <- as.data.frame(X[-folds[[i]],])
  test_df_lda <- as.data.frame(X[folds[[i]],])

  colnames(train_df_lda)[31] <- "y_lda"
```

```
  colnames(test_df_lda)[31] <- "y_lda"

  #training
  model <- lda(y_lda~X2+X9+X10+X17+X20+X23, data = train_df_lda)

  #predict
  lda.pred <- predict(model, newdata = test_df_lda)

  #error_rate
  error_rates_lda[i, ] <- length(which(as.vector(lda.pred$class) != as.vector(test_df_lda$y_lda)))/leng
}

print(mean(as.vector(error_rates_lda))) # tout 0.22 que les xi 0.2
```

```
## [1] 0.2
```

Ici, nous avons appliqué la fonction QDA sur tous les variables, et l'erreur est 0.22, en même temps nous avons pris les mêmes données en appiquant le QDA seulement sur les xi plus significatifs, l'erreur est 0.2. Nous trouvons que le LDA fonctionne mieux s'il est appliqué sur les variables sélectionnés.

Mais nous ne savons pas si l'analyse discriminante lineaire suffisant ou pas, car ici nous suppons que la matrice de variance est commune pour toutes les classes(hypothese d'homoscedasticite).
Nous devons vérifier si c'est le cas.

```
# nous calculons l'erreur moyenne entre les éléments des matrices de variance pour les deux classes
sum(abs(var(X_class1) - var(X_class2))) / 30
```

```
## [1] 3.531246
```

Mais nous ne sommes pas sur que l'hypothese d'homoscedasticite est correct si nous ne faisons pas un meilleur test. Si on fait le QDA:

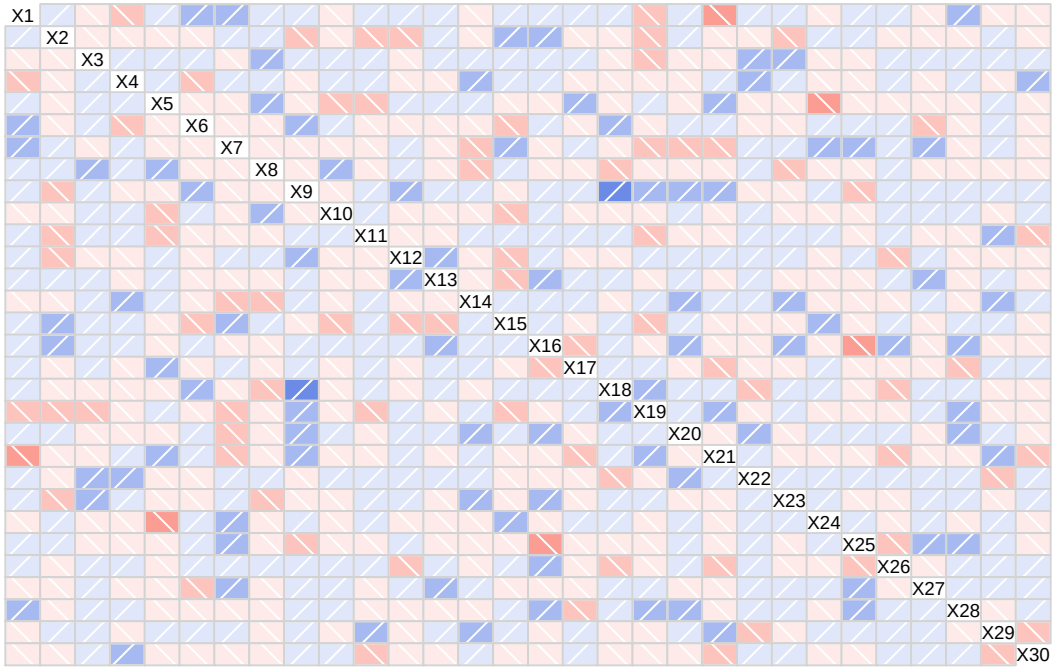
```
error_rates = matrix(nrow=10, ncol=1)
# using 10-folds cross-validation
for (i in 1:10) {

  # make the folds
  train_df_qda = as.data.frame(X[-folds[[i]],])
  test_df_qda = as.data.frame(X[folds[[i]],])
  colnames(train_df_qda)[31] = "y_qda"
  colnames(test_df_qda)[31] = "y_qda"

  # training
  model = qda(y_qda~X2+X9+X10+X17+X20+X23 , data = train_df_qda)

  # predict
  qda.pred = predict(model, newdata = test_df_qda)

  # error rate
  error_rates[i,] = length(which(
    as.vector(qda.pred$class) != as.vector(test_df_qda$y_qda)
  ))/length(as.vector(test_df_qda$y_qda))
}
mean(as.vector(error_rates))   # 1.Pour tous les vbs 0.295  2.que les vbs significatifs 0.235
```

```
## [1] 0.235
```

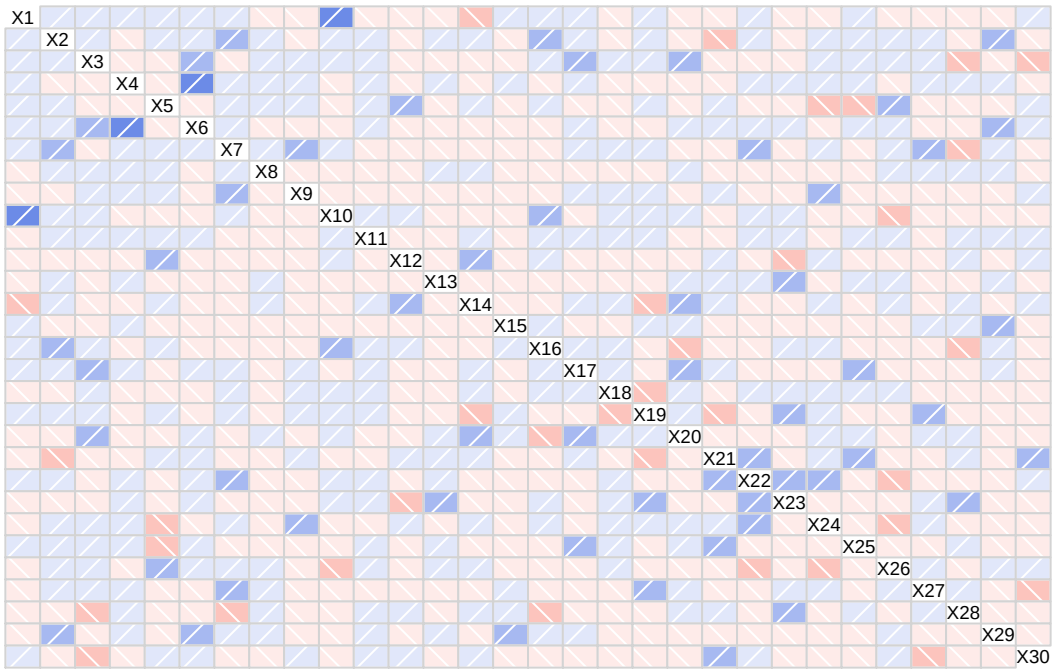
Ici, nous avons appliqué la fonction QDA sur tous les variables, et l'erreur est 0.295, en même temps nous avons pris les mêmes données en appiquant le QDA seulement sur les xi plus significatifs, l'erreur est 0.235. Le QDA aussi, il fonctionne mieux avec les xi sélectionnés.

Nous allons étudier l'indépendance des variables, s'ils sont indépendants, nous pouvons appliquer le classifieur bayesien naif.

```
corrgram(X_class1)
```




```
corrgram(X_class2)
```



Nous pouvons voir le graphique, cette hypothèse semble clairement non recevable car les variables sont corrélés. Dans ce cas là, nous pouvons appliquer le scale sur les variables. Pour réduire le taille de rapport, ici nous rendrons pas le détail, d'après résultat, nous trouvons que c'est toujours pas le cas pour appliquer le classifieur bayesien naif.

```
SX_class1 = scale(as.matrix(X[X$y == 1, 1:30]))
SX_class2 = scale(as.matrix(X[X$y == 2, 1:30]))

corrgram(SX_class1)
```

```
corrgram(SX_class2)
```

Deuxième méthode: Utiliser une forêt aléatoire de décision binaire

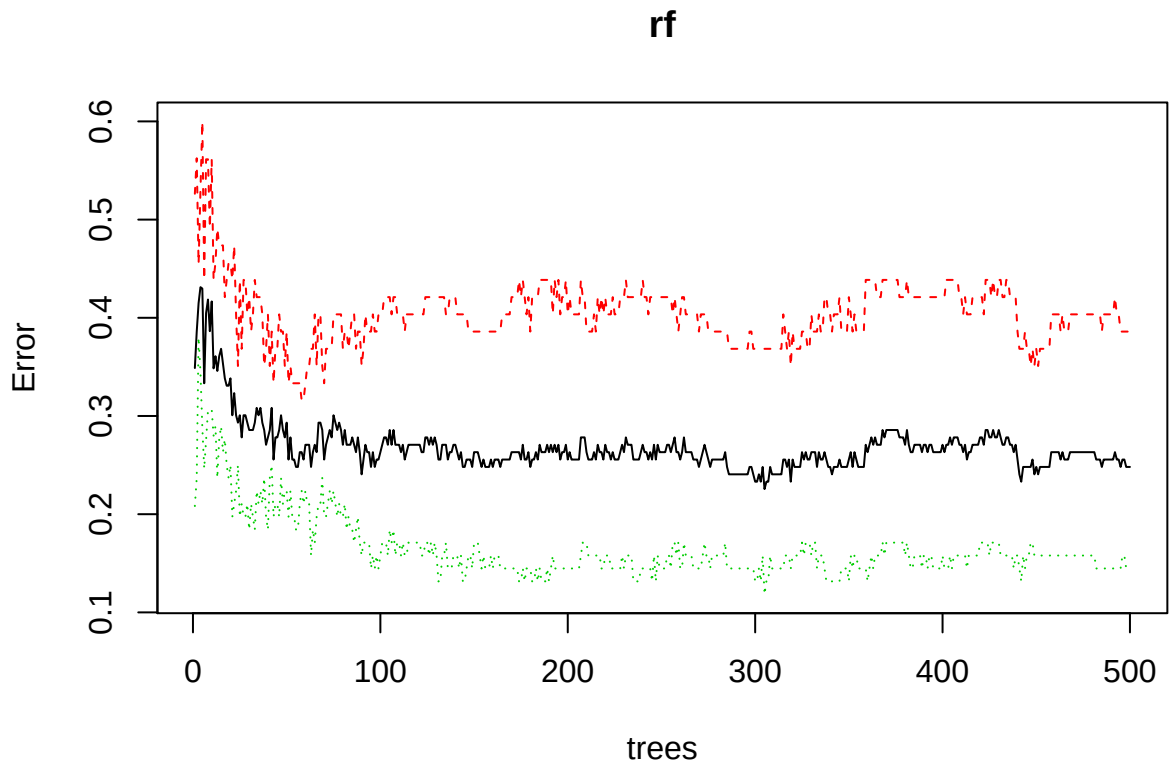
Pour réduire l'erreur sur les données, nous utilisons une foret aléatoire d'arbres de decision.

```
library("randomForest")
```

```
## Warning: package 'randomForest' was built under R version 3.3.3
## randomForest 4.6-12
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
## The following object is masked from 'package:ggplot2':
##
```

```
##     margin
```

```
#X <- read.table('tp3_clas_app.txt', header = TRUE)
X$y <- as.factor(X$y)

res <- split_dataset(X, 2 / 3)#nous utilisons la fonction définise avant
train_set <- res$train_set
test_set <- res$test_set

# Creation magique (et naive) de la foret
rf <- randomForest(y ~ ., data = train_set)

plot(rf)
```




```
# Analyse de l'erreur en fonction de la taille des arbres et du nombre d'arbres
#p = dim(train_set)[2]#le nombre de colonnes
#mtry = floor(sqrt(p)) # valeur empirique optimale pour un pbm de classification
#rfopt <- randomForest(y ~ ., data = train_set, ntree = 50, mtry = mtry)
#ntree = 50 selon le graphique
predictions <- predict(rf, newdata = test_set[, 1:30], type = 'class')
erreur <- mean(predictions != test_set$y)
erreur  #  0.1791045
```

```
## [1] 0.1791045
```

Nous observons un résultat relativement satisfaisant (*erreur = 0.1791045*).
Celui-ci est meilleur car il peut réduire le Test MSE, et est facilement et rapidement interprétable.

Troisieme idée: regression logistique avec validation croisée

Nous pouvons utiliser la regression logistique pour estimer directement les probabilites d'appartenance à la classe.
Si nous utilisons seulement la methode de l'ensemble de validation, nous avons obtenu une erreur proche de 0.5, c'est pas bon du tout et c'est pire qu'un modèle de bernouilli.
Afin de réduire l'erreur, nous utilisons la methode de validation croisée(Cross-Validation) qui consiste à réechantionner plusieurs l'ensemble d'apprentissage pour garder uniquement le meilleur modèle.

```
library('DAAG')
```

```
## Warning: package 'DAAG' was built under R version 3.3.3
```

```
##
## Attaching package: 'DAAG'
```

```
## The following object is masked from 'package:MASS':
##
##     hills
```

```
#X <- read.table('tp3_clas_app.txt', header = TRUE)
X$y <- as.factor(X$y)
obj <- glm(y ~ ., data=X, family = binomial(logit))
CVbinary(obj, nfolds=10)
```

```
##
## Fold:  7 1 6 10 9 3 4 2 8 5
## Internal estimate of accuracy = 0.865
## Cross-validation estimate of accuracy = 0.725
```

Selon le résultat, l'estimation de la précision par la méthode de la validation croisée avec un modele de regression logistique binaire est de *0.77*. Donc, l'erreur = 1-0.75 = 0.25.

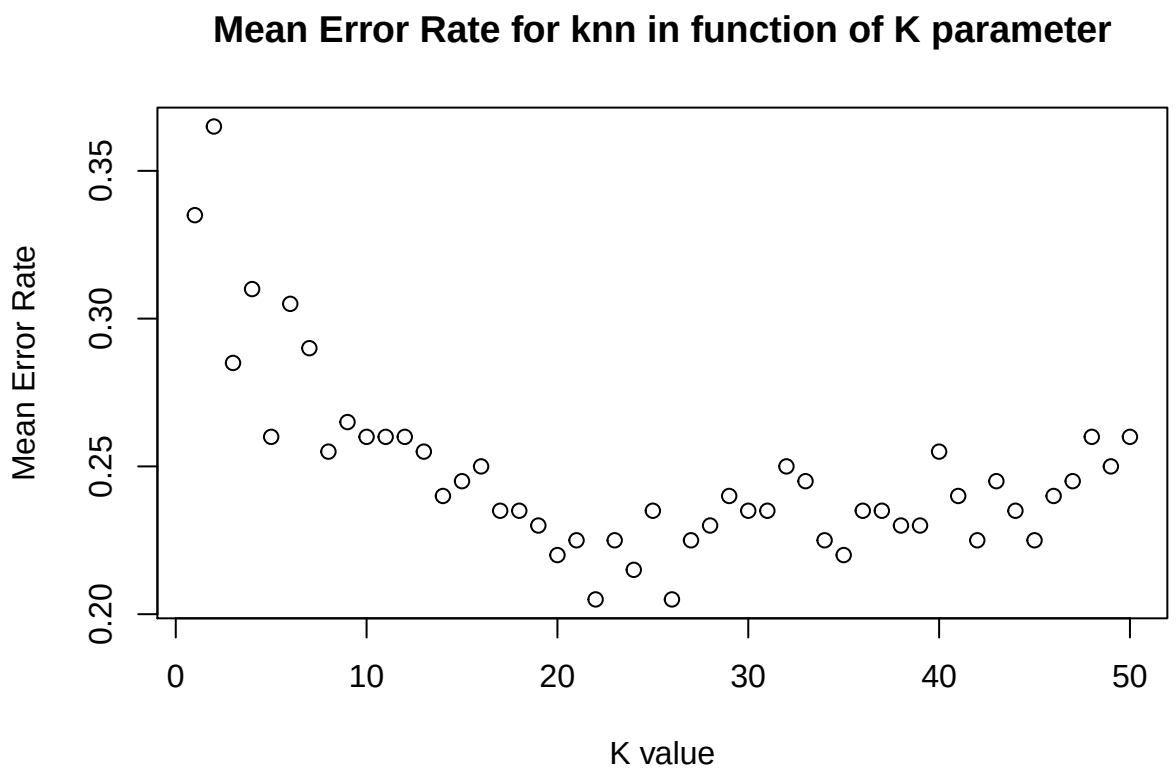
Quatième méthode : KNN

```
library(class)
error_rates <- matrix(nrow = 50, ncol = 10)

for(k in 1:50){
  #using 10-folds cross-valisation
  for (i in 1:10) {
    #make the folds
    train_df <- as.data.frame(X[-folds[[i]],])
    test_df <- as.data.frame(X[folds[[i]],])
    colnames(train_df)[31] = "y_df"
    colnames(test_df)[31] = "y_df"

    #knn prediction
    knn.pred <- knn(train_df[, -31], test_df[, -31], train_df$y_df, k = k)

    error_rates[k,i] <-
      (length(which(as.vector(knn.pred) !=
                      as.vector(test_df$y_df)))/length(test_df$y_df))
  }
}
#last prediction confusion table
```

```r
table(knn.pred, test_df$y_df)
```

```
##
## knn.pred  1  2
##        1  2  0
##        2  3 15
```

```r
mean_error_rates <- apply(error_rates, 1, mean)

plot(1:50, mean_error_rates,
     main = "Mean Error Rate for knn in function of K parameter",
     xlab = "K value", ylab = "Mean Error Rate")
```




```r
mean(mean_error_rates)
```

```
## [1] 0.2477
```

Selon le résultat, nous trouvons que l'erreur moyen égale 0.2477.
Si nous comptons tous les résultats que nous avons obtenu avant:

Méthode	Erreur
QDA(prendre tous les xi)	0.295
Logitique regression	0.250
KNN	0.247
QDA(prendre que les xi sélectionné)	0.235
LDA(prendre tous les xi)	0.220
LDA(prendre que les xi sélectionné)	0.200

Méthode	Erreur
Random Forest	0.179

D'apres le tableau, nous voyons que le meilleur méthode est Random Forest.

Regression

Penser à changer le forme de Xi

Si nous considérons à construire une base de données qui contient non seulement Xi, mais aussi X^2, log(X), et puis nous utilisons le Forward Selection pour voir si ce méthode fonnctionne bien

```
Ycarre = data_reg[,-51]^2
Ylog10 = log10(data_reg[,-51])
Ylog2 = log2(data_reg[,-51])
```

```
newdata_reg = data.frame(data_reg[,-51], Ycarre, Ylog10, Ylog2,data_reg[,51])
colnames(newdata_reg[201]) ="y"
```

```
min.model = lm(newdata_reg[,201] ~ 1, data=newdata_reg)
biggest <- formula(lm(newdata_reg[,201]~-newdata_reg[,201],newdata_reg))
biggest
```

```
## newdata_reg[, 201] ~ -newdata_reg[, 201]
```

```
fwd.model = step(min.model, direction='forward', scope=biggest)
```

```
## Start:  AIC=1124.64
## newdata_reg[, 201] ~ 1
```

Partition des donnés

La première étape de traitement des donnés consiste à réaliser une partition entre donnés d'apprentissage (2/3) et donnés de test (1/3). Nous choisissons ces proportions, qui sont couramment utilisés. Ainsi, seules les donnés d'apprentissage seront utilisés pour regression, tandis que les donnés de test serviront à aluer la performance du modèle.

Subset selection

Nous commencons par définir la fonction qui nous permettra d'extraire les meilleurs sous-ensembles de préicteurs. Cette fonction est appliqué au donnés d'apprentissage.

```
getSubsets <- function(formula, train_data, main = '', xlab = '', ylab = ''){
  reg.fit_only_means = regsubsets(formula, data = train_data, really.big = TRUE)
  pdf("./pdf/adjr2.pdf")
  plot(reg.fit_only_means, scale = "adjr2", main = main, xlab = xlab, ylab = ylab + ' adjr2')
  dev.off()
  pdf("./pdf/bic.pdf")
  plot(reg.fit_only_means, scale = "bic", main = main, xlab = xlab, ylab = ylab + ' bic')
  dev.off()
  pdf("./pdf/Cp.pdf")
  plot(reg.fit_only_means, scale = "Cp", main = main, xlab = xlab, ylab = ylab + ' Cp')
```

```
  dev.off()
  summary.out <- summary(reg.fit_only_means)
  as.data.frame(summary.out$outmat)
  return(reg.fit_only_means)
}
wholedata.subsets <- getSubsets(y~., data_train,
                                'Subset Selection for 2/3rds of whole set',
                                'Predictors', 'Values')
```

Ici, nous gardons les graphiques dans l'archivre. D'aprè ces 4 graphiques, les principaux préicteurs sont : X_4, X_{19}, X_{22}, X_{24}, X_{27}, X_{35}, X_{39}. Nous les conservons pour la suite. Notre modele sera donc du type :

$Y = \beta_0 + \beta_1 X_4 + \beta_2 X_{19} + \beta_3 X_{22} + \beta_4 X_{24} + \beta_5 X_{27} + \beta_6 X_{35} + \beta_7 X_{39} + \beta_8 X_{41}$

Cross Validation

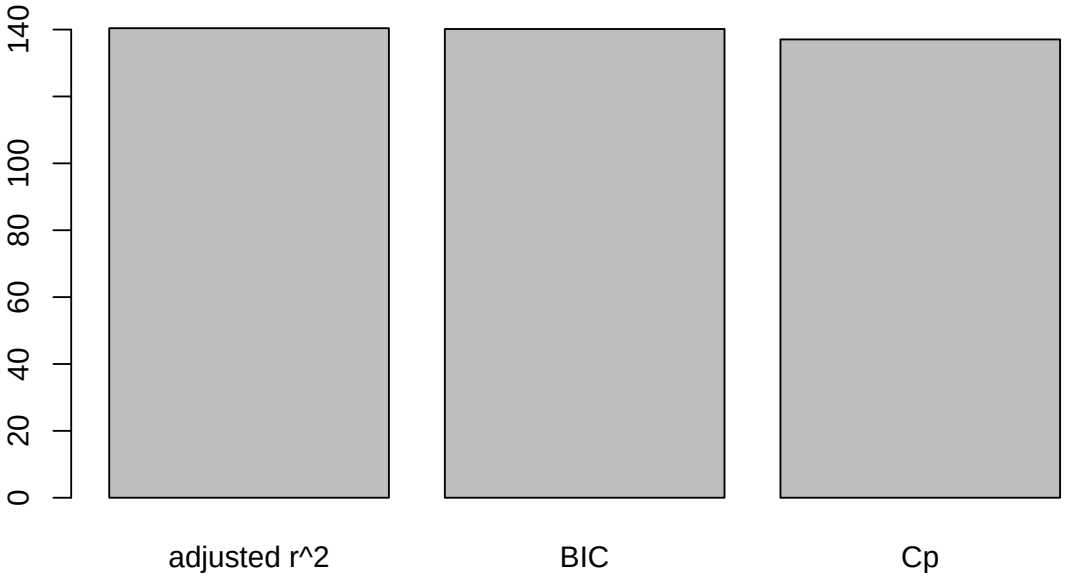
Nous faisons le choix d'utiliser le k-fold cross Validation, qui a prouvé son efficacit? empirique avec un bon compromis bias-variance. Dans le cadre de ce TP, nous optons pour k = 10. Nous divisons donc les donnees en k parties égales. Pour chacun on fit un modele sur tousles autres plis. Enfin, nous calculons le R^2 de ce modele. La fonction de 10-Fold Cross Validation est définie comme suit :

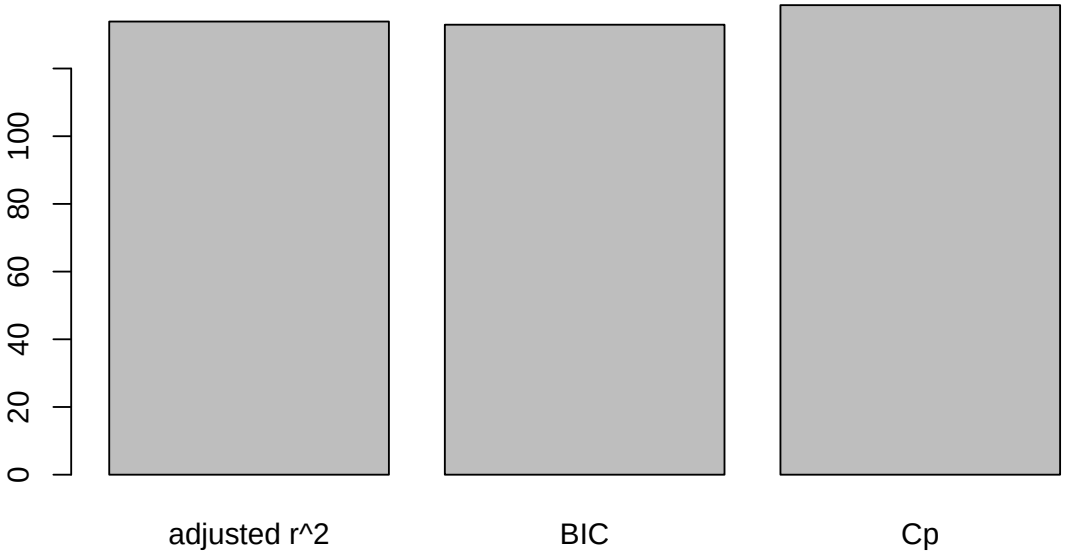
```
crossValidate <-function(formula, data, k = 10){
n <- nrow(data)
  folds <- sample(1:k, n, replace = TRUE)
  cv <- 0
  for(k in 1:k){
    reg <- lm(formula, data = data[folds != k,])
    pred <- predict(reg, newdata = data[folds == k,])
    cv <- cv + sum((data$y[folds == k] - pred)^2)
  }
  cv <- cv/n
  return(cv)
}
```

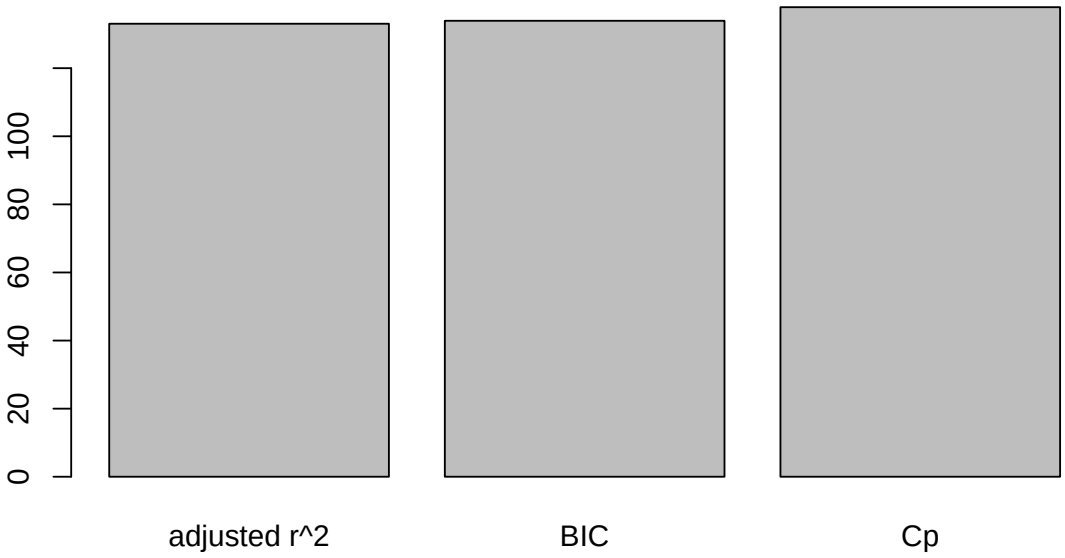
La fonction BatchCrossValidate nous permet d'appliquer le cross validation ? tous les modeles trouvé lors de la phase.
D'après le graphique avant, nous avons trouvé les xi qui sont plus significatifs: X4, X19, X22, X24, X27, X35, X39.

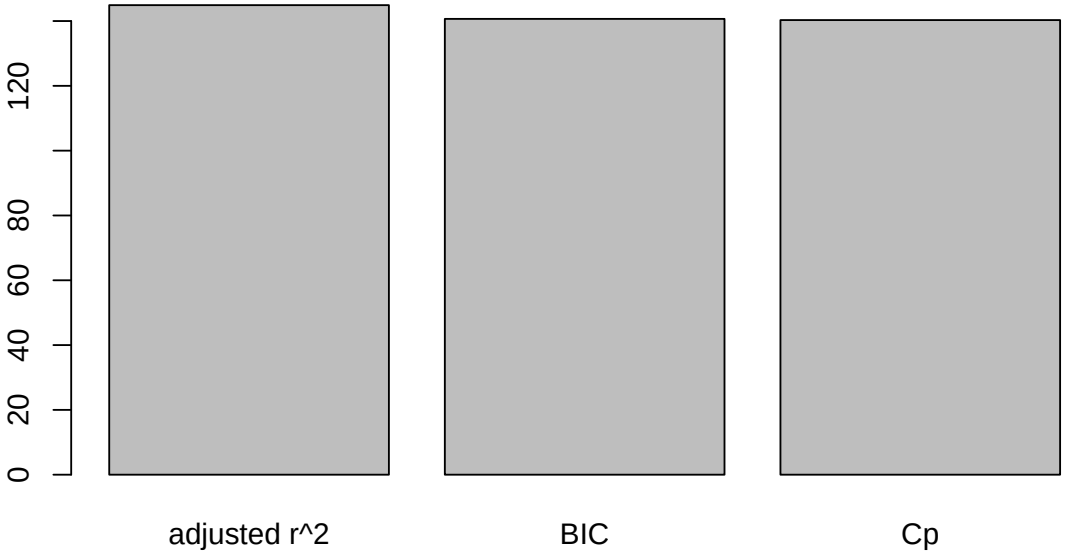
En l'executant 10 fois, on obtient un tableau recupitulatif de la cross-validation de chaque methode :

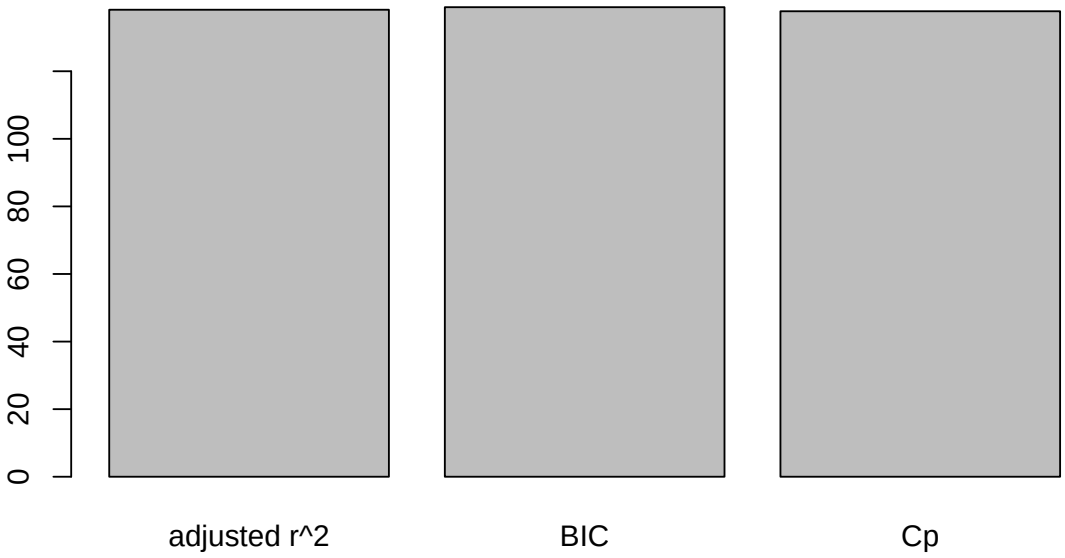
```
resultat = plot_cross_validations(5)  #find the moyen pour les 4 cols
```

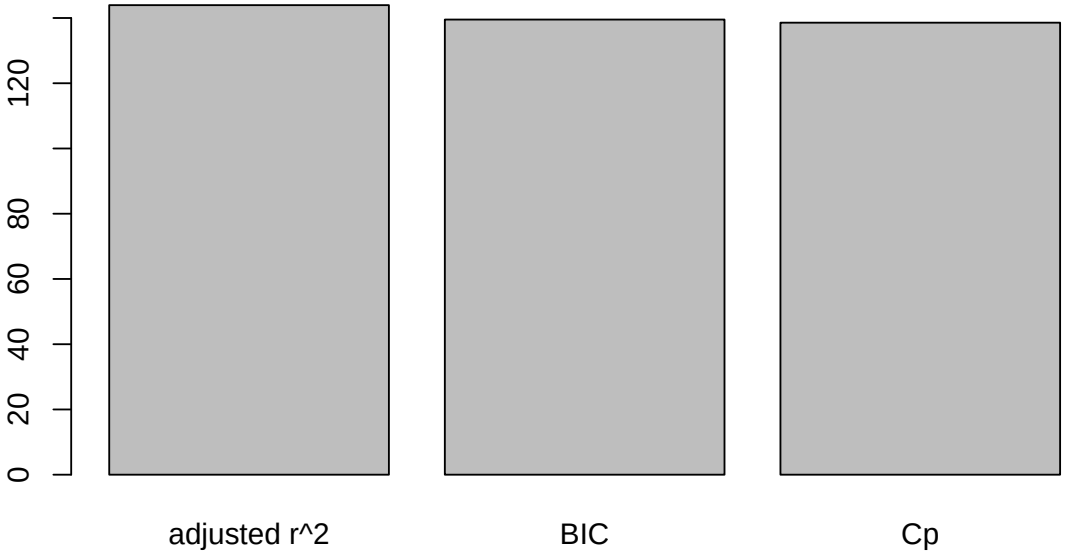

0
20
40
60
80
100
140
adjusted r^2
BIC
Cp

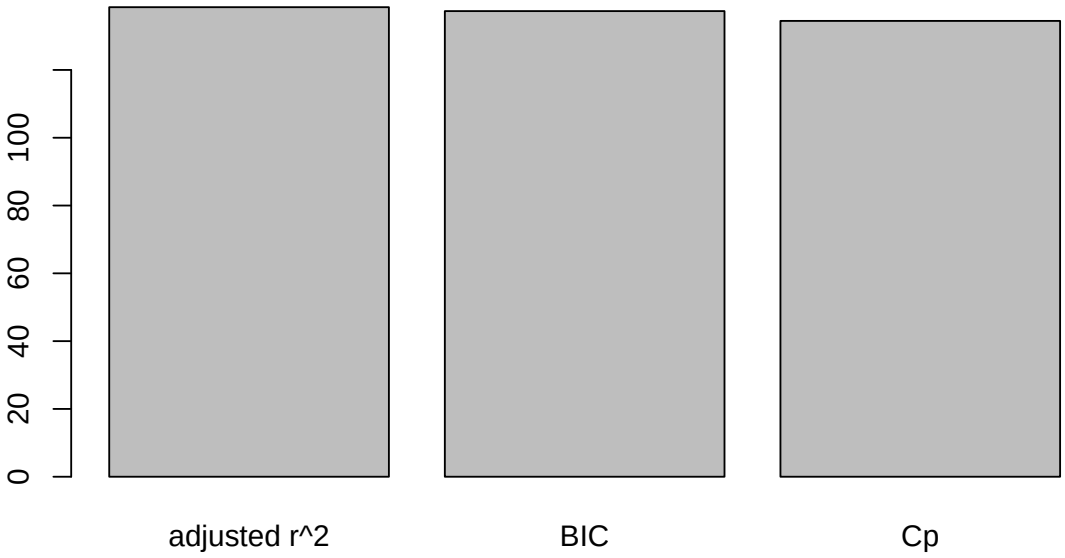

0
20
40
60
80
100
adjusted r^2
BIC
Cp

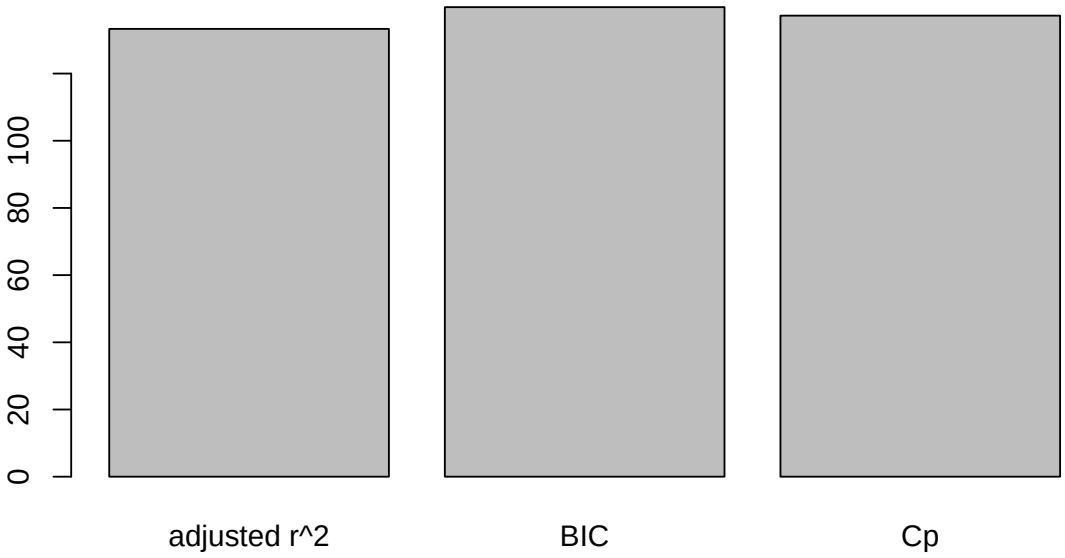

0
20
40
60
80
100
adjusted r^2
BIC
Cp

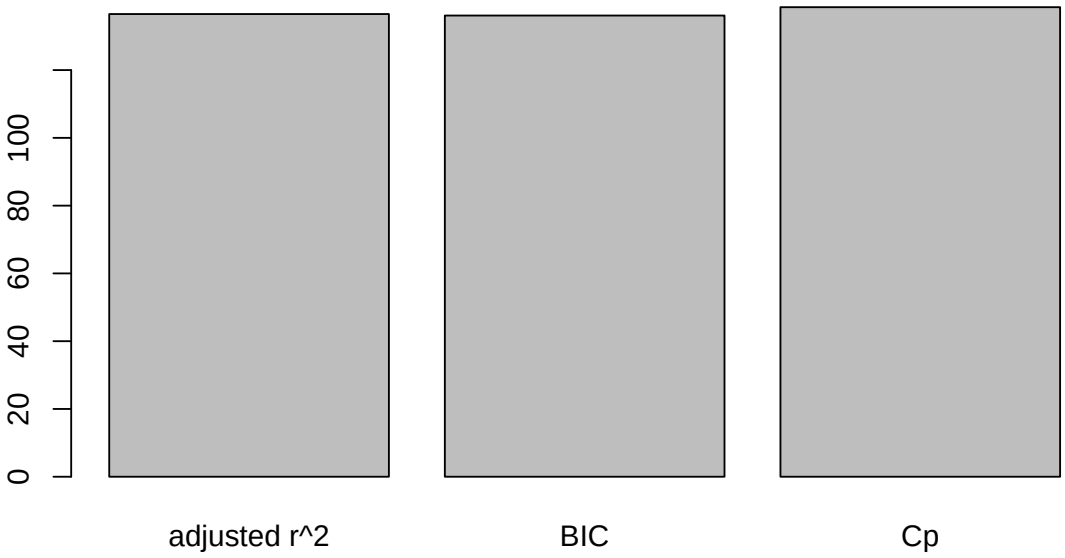

0
20
40
60
80
120
adjusted r^2
BIC
Cp

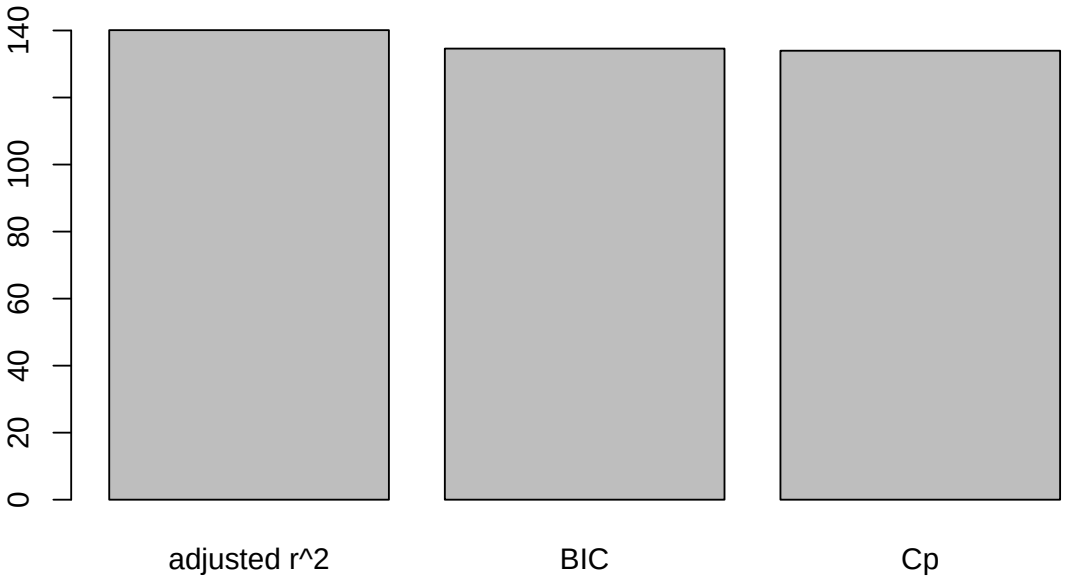

0
20
40
60
80
100
adjusted r^2
BIC
Cp


0
20
40
60
80
120
adjusted r^2
BIC
Cp


0
20
40
60
80
100
adjusted r^2
BIC
Cp


0
20
40
60
80
100
adjusted r^2
BIC
Cp






```
print(mean(resultat$`adjusted r^2`))
```

```
## [1] 126.6199
```

```r
print(mean(resultat$bic))
```

```
## [1] 125.8157
```

```r
print(mean(resultat$cp))
```

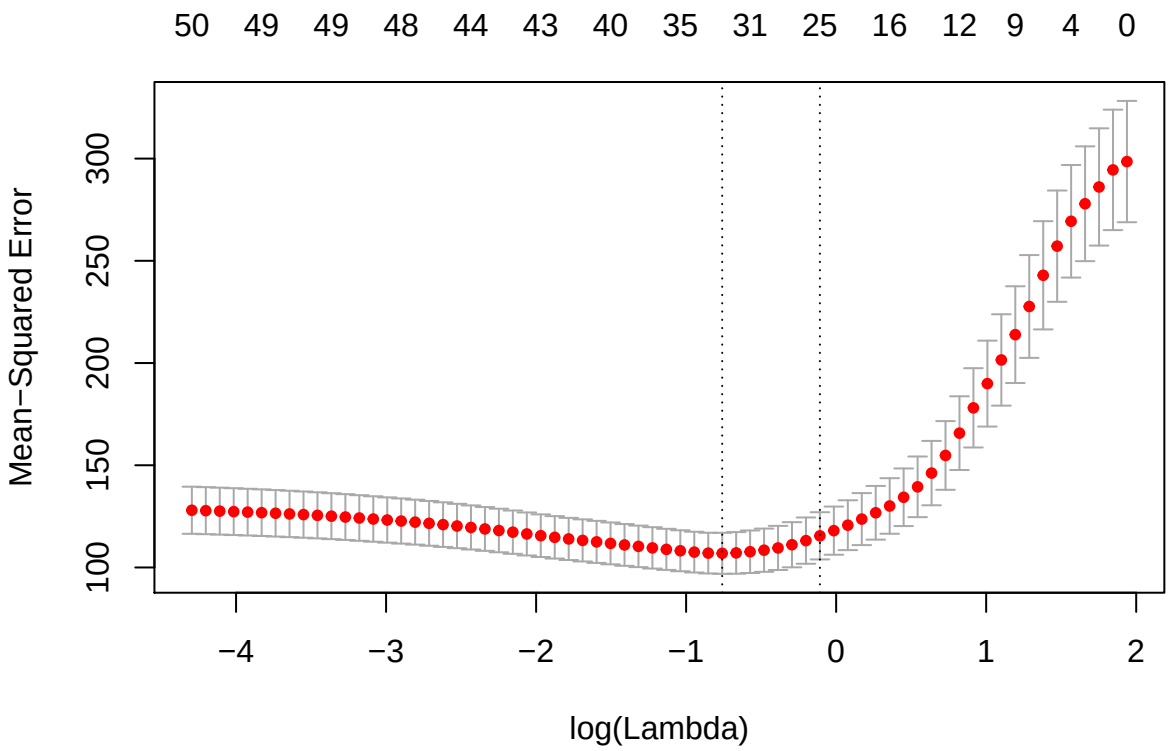
```
## [1] 125.8647
```

Regularisation

```r
x <- model.matrix(y~., data_reg)
y<-as.data.frame(data_reg$y)

set.seed(1)
xapp <- x[train, ]
yapp <- y[train, ]
xtst <- x[-train, ]
ytst <- y[-train, ]
```

Lasso

```r
cv.out_lasso <- cv.glmnet(xapp,yapp,alpha=1)

plot(cv.out_lasso)
```



```
bestlam_lasso=cv.out_lasso$lambda.min
fit.lasso<-glmnet(xapp,yapp,lambda = bestlam_lasso,alpha = 1)
lasso.pred<-predict(fit.lasso, s = cv.out_lasso$lambda.min, newx = xtst)

#calcul MSE
print(mean((ytst-lasso.pred)^2))
```

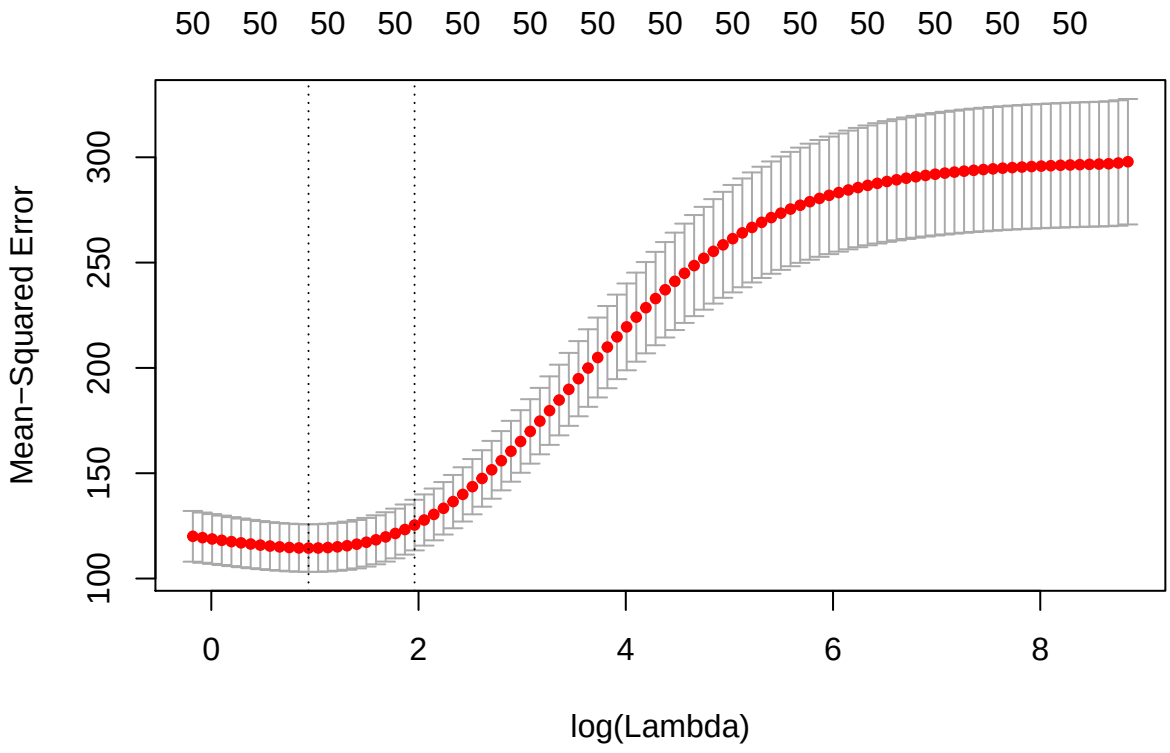
```
## [1] 79.70625
```

Ridge

```
library(glmnet)
cv.out_ridge <- cv.glmnet(xapp, yapp, alpha = 0,parallel=TRUE)
```

```
## Warning: executing %dopar% sequentially: no parallel backend registered
```

```
plot(cv.out_ridge)
```




```
bestlam_ridge=cv.out_ridge$lambda.min
fit.ridge<-glmnet(xapp,yapp,lambda = bestlam_ridge,alpha = 0)
ridge.pred <- predict(fit.ridge, s = cv.out_ridge$lambda.min, newx = xtst)
summary(ridge.pred)
```

```
##        1
##  Min.   :-50.974
##  1st Qu.:-15.209
##  Median : -7.990
```

```
##  Mean   : -8.283
##  3rd Qu.: -2.689
##  Max.   : 24.595
```

```
#calcul de MSE
print(mean((ytst-ridge.pred)^2))
```

```
## [1] 94.21217
```

Si nous comptons tous les résultats que nous avons obtenu avant:

Méthode	Test Erreur
Adjusted r^2	126.6199
Cp	125.8647
BIC	125.8157
Ridge	94.21217
lasso	79.70625

Nous voyons bien que le 'lasso' est le meilleur!