BE3 - partie 1

Tulio NAVARRO TUTUI, Filipe PENNA CERAVOLO SOARES

06 December, 2022

On reprend le jeu de données concernant la valeur des logements des villes aux alentours de Boston. On cherche à identifier un bon modèle part régression pénalisée, CART, Boosting et Random Forest.

Les variables utilisées sont les suivantes:

CRIM taux de criminalité par habitant

ZN proportion de terrains résidentiels

INDUS proportion de terrains industriels

CHAS 1 si ville en bordure de la rivière Charles 0 sinon

NOX concentration en oxydes d'azote

RM nombre moyen de pièces par logement

AGE proportion de logements construit avant 1940

DIS distance du centre de Boston

RAD accessibilité aux autoroutes de contournement

TAX taux de l'impôt foncier

PTRATIO rapport élèves-enseignant par ville

LSTAT % de la population 'a faibles revenus

class valeur du logement en 1000$

L'objectif de ce BE est de comparer différents modèles de machines learning : le modèle linéaire, le modèle linéaire pénalisé (Ridge et Lasso), les modèles linéaires de réduction de dimension (PCR et PLS), les modèles à base d'arbres (CART, Boosting, Bagging et Random Forest).

Lasso = L1
Ridge = L2

Modèles linéaires

On commence par ajuster un modèle linéaire sans et avec interactions à partir de l'ensemble d'apprentissage constitué de 300 observations. Evaluer la qualité prédictive de ces deux modèles sur l'ensemble test constitué de 206 observations. On pourra utiliser la commande RMSE du package DiceEval.

```
library(DiceEval)
```

```
## Loading required package: DiceKriging
```

```
library(car)
```

```
## Loading required package: carData
```

```
library(MASS)

housing = read.table("housing_new.txt", header = T)
p = ncol(housing)
summary(housing)
```

```
##       CRIM                ZN              INDUS            CHAS
##  Min.   : 0.0060   Min.   :  0.00   Min.   : 0.46   Min.   :0.00000
##  1st Qu.: 0.0820   1st Qu.:  0.00   1st Qu.: 5.19   1st Qu.:0.00000
##  Median : 0.2565   Median :  0.00   Median : 9.69   Median :0.00000
##  Mean   : 3.6135   Mean   : 11.36   Mean   :11.14   Mean   :0.06917
##  3rd Qu.: 3.6770   3rd Qu.: 12.50   3rd Qu.:18.10   3rd Qu.:0.00000
##  Max.   :88.9760   Max.   :100.00   Max.   :27.74   Max.   :1.00000
##       NOX               RM              AGE              DIS
##  Min.   :0.3850   Min.   :3.561   Min.   :  2.90   Min.   : 1.130
##  1st Qu.:0.4490   1st Qu.:5.886   1st Qu.: 45.02   1st Qu.: 2.100
##  Median :0.5380   Median :6.208   Median : 77.50   Median : 3.208
##  Mean   :0.5547   Mean   :6.285   Mean   : 68.57   Mean   : 3.795
##  3rd Qu.:0.6240   3rd Qu.:6.623   3rd Qu.: 94.08   3rd Qu.: 5.189
##  Max.   :0.8710   Max.   :8.780   Max.   :100.00   Max.   :12.126
##       RAD              TAX           PTRATIO          LSTAT
##  Min.   : 1.000   Min.   :187.0   Min.   :12.60   Min.   : 1.73
##  1st Qu.: 4.000   1st Qu.:279.0   1st Qu.:17.40   1st Qu.: 6.95
##  Median : 5.000   Median :330.0   Median :19.05   Median :11.36
##  Mean   : 9.549   Mean   :408.2   Mean   :18.46   Mean   :12.65
##  3rd Qu.:24.000   3rd Qu.:666.0   3rd Qu.:20.20   3rd Qu.:16.95
##  Max.   :24.000   Max.   :711.0   Max.   :22.00   Max.   :37.97
##      class
##  Min.   : 5.00
##  1st Qu.:17.02
##  Median :21.20
##  Mean   :22.53
##  3rd Qu.:25.00
##  Max.   :50.00
```

```
dim(housing)
```

```
## [1] 506  13
```

```
set.seed(23)
u = sample(1:nrow(housing), 300)
housing.train = housing[u,]
housing.test = housing[-u,]
```

```
library(caret)
```

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

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

```
##
## Attaching package: 'caret'
```

```
## The following objects are masked from 'package:DiceEval':
##
##     MAE, R2, RMSE
```

```
# Ajuster un modèle linéaire simple sans interactions
mod_lm <- lm(class ~., data = housing.train)
Y_pred_lm = predict(mod_lm, housing.test[, -p]) -p = excluir a coluna de previsão
RMSE_lm = RMSE(housing.test[, p], Y_pred_lm)  # root mean square error
RMSE_lm # 5.9          p = pegar os dados de previsão
```

```
## [1] 5.900007
```

```
# Ajuster un modèle linéaire avec interactions
#a completer :              todos os parametros e suas iterações duplas
mod_lm_inter <- lm(class ~.^2, data = housing.train)
length(na.omit(mod_lm_inter$coefficients))
```

```
## [1] 79
```

```
#a completer :
Y_pred_lm_inter = predict(mod_lm_inter, housing.test[, -p])
#a completer :
RMSE_lm_inter = RMSE(housing.test[, p], Y_pred_lm_inter)
RMSE_lm_inter  # 4.41
```

```
## [1] 4.416989
```

On remarque qu'en ajoutant les termes d'interaction l'erreur a diminué. Cependant le modèle avec interaction est très complexe (79 termes) donc certainement trop variable. L'idée est de chercher un modèle plus prédictif, i.e. avec une RMSE plus faible que 4.836.

Modèles linéaires pénalisés

Ajuster une régression lasso (commande lars du package lars). Pour ce faire, il est nécessaire de créer une matrice contenant tous les termes (variables principales et les interactions).

```
#-----------------------------------------------------------------
# lasso
#-----------------------------------------------------------------

library(lars)
```

```
## Loaded lars 1.3
```

```
y = as.matrix(housing.train[, p])
x = as.matrix(housing.train[, -p])
x_extend = x

# makes the binary combinations of all columns
for (i in 1:(p-2)) {
  for (j in 2:(p-1)) {
    x_extend = cbind(x_extend, x[, i] * x[, j])
  }
}

mod_lasso = lars(x_extend, y, type = "lasso")
```

lambda ~ 1/(soma dos coefficientes)
t >= soma do modulo dos coeficientes = penalidade

— Etudier la matrice des coefficients (commande coef.lars) et tracer le modèle (commande plot).

```
dim(coef.lars(mod_lasso)) # 186 valeurs de lambda différentes et 133 coefficients
```

coeff = beta

```
## [1] 186 133
```

soma dos coefs é maior
t é maior

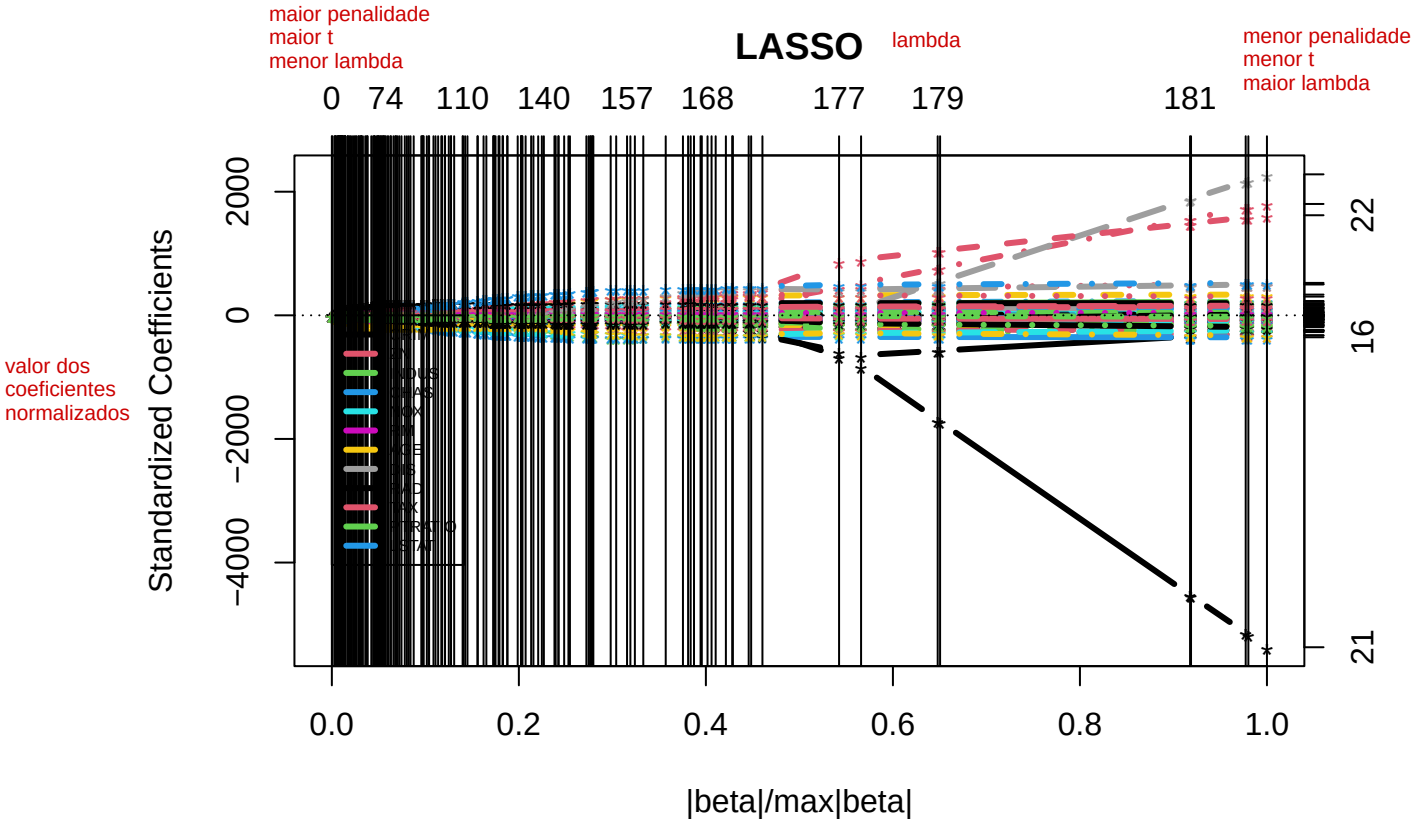
```
# Faire un choix de ligne entre 1 (très forte pénalité) et 186 (faible pénalité)
l = 10
sum(coef.lars(mod_lasso)[l, ] == 0)  # n de coefs = 0
```

soma dos coefs é menor
t é menor

```
## [1] 127
```

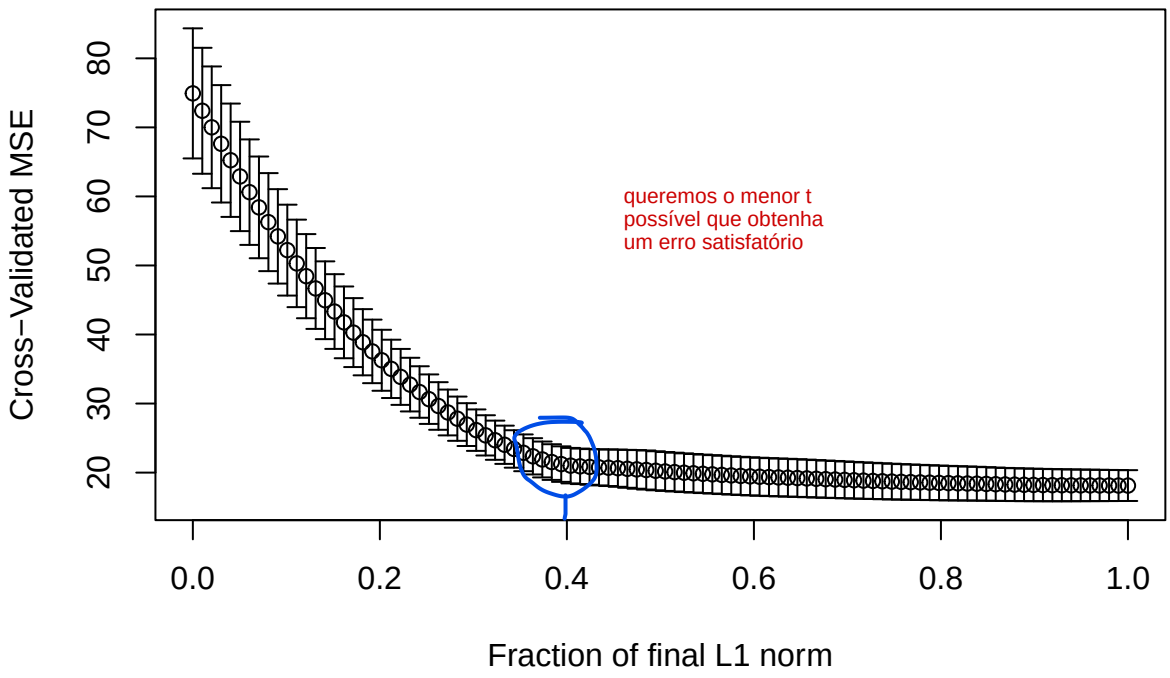
With l = 10, sum equals 127, whareas with l = 100, sum equals 82. This shows that with a lesser l, more coefficients are getting 0 as a value.

```
par(mfrow = c(1, 1))
plot(mod_lasso, cex = 1, lwd = 3, col = 1:(p-1))
legend(
  0,
  0,
  names(housing.train[, -p]),
  col = 1:(p-1),
  lty = rep(1, p-1),
  lwd = rep(3, p-1),
  cex = 0.5
)
```



— Faire un choix du meilleur modèle en étudiant la décroissance de l'erreur par validation croisée (commande cv.lars). Faire un choix de la meilleure fraction (directement liée à la valeur du lambda. Fraction = 1 (resp. Fraction = 0) correspond à $\lambda = 0$ (resp. $\lambda = +\infty$)).

```
CV = cv.lars(x, y, K = 10, index=seq(from = 0, to = 1, length = 100),
             trace = FALSE, plot.it = TRUE, se = TRUE, type = "lasso", mode = "fraction")
```




```
#a completer :
value = 0.4
```

- Donner le nombre de coefficients non nuls du modèle final retenu. La pénalité a-t-elle jouer son rôle ?

```
#a completer :
coef_lasso_beter <- predict.lars(mod_lasso, s = value, mode = "frac", type = "coef")
#a completer :
nb_coef_nul = sum(coef_lasso_beter$coefficients == 0)
nb_coef_nul
```

```
## [1] 51
```

Obtemos 51 coeficientes nulos. Um valor menor que os valores que obtivemos antes. L portanto seria maior que 100.

— Evaluer la qualité prédictive du modèle sur l'ensemble test (commande predict.lars).

```
newx = as.matrix(housing.test[,-p])
for (i in 1:(p-2))
{
  for (j in 2:(p-1))
  {
    newx = cbind(newx,newx[,i]*newx[,j])
  }
```

```
}
#a completer :
fits <- predict.lars(mod_lasso, newx, s = value, mode = "frac")
#a completer :
RMSE_lasso = RMSE(housing.test[, p], fits$fit)
c(RMSE_lm_inter, RMSE_lasso)
```

```
## [1] 4.416989 4.581404
```

Ajuster maintenant une régression Ridge. La pénalité Ridge ne s'applique pas sur l'intercept. Il est donc nécessaire d'enlever le terme constant avant d'appliquer la procédure lm.ridge de la library MASS.

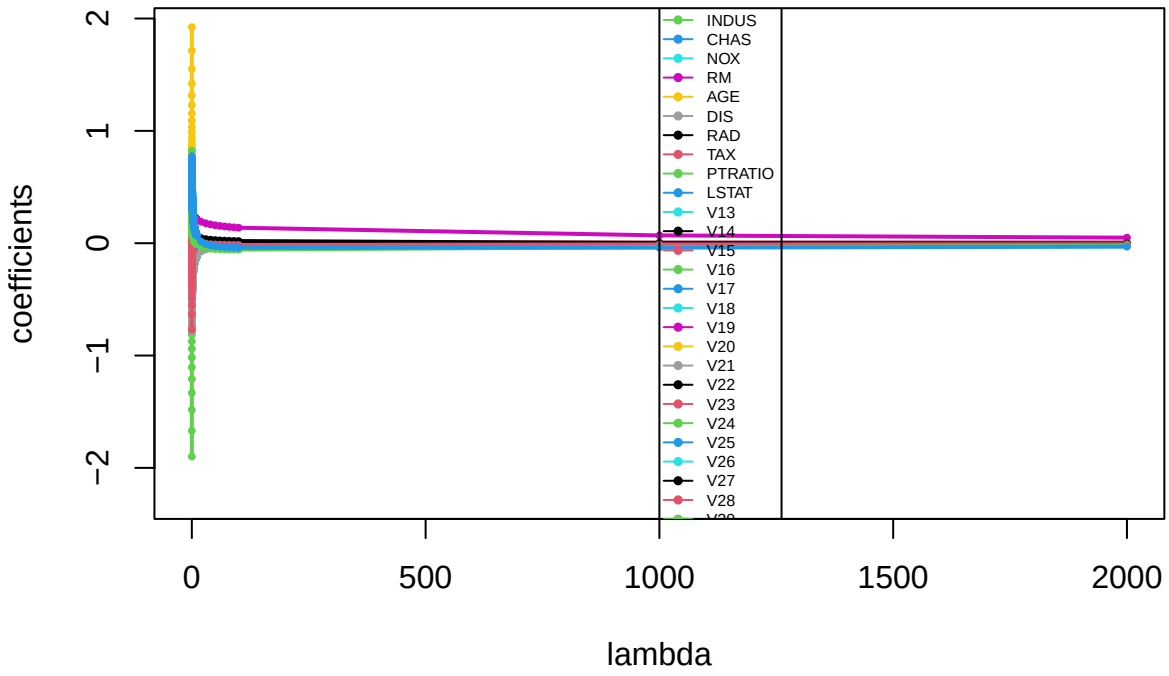
```
#----------------------------------------
#ajustement ridge
#----------------------------------------
library(MASS)
#on part d'une matrice centrée réduite
housing.train2 = as.data.frame(scale(cbind(x_extend, y))) # scale does the normalization of columns
#on garde en mémoire moyenne et écart type des données initiales
mean.train = apply(cbind(x_extend, y), 2, mean)  # 2 = mean is by column
sd.train = apply(cbind(x_extend, y), 2, sd)  # 2 = sd is by column

#On se donne un vecteur de poids lambda
lambda = c(seq(0.01, 10, 0.01), seq(10, 100, 10), 1000, 2000)

p2 = ncol(housing.train2)
mod_ridge = lm.ridge(V134 ~.-1, data = housing.train2, lambda = lambda)
# V134 is the class column with centre reduit
```

- Tracer les différents beta en fonction du λ. Que remarquez-vous ?

```
#tracer des différentes valeurs des beta en fonction de lambda
beta_mat = mod_ridge$coef
par(mfrow = c(1, 1))
plot(lambda, beta_mat[1, ], ylim = c(min(beta_mat),
     max(beta_mat)), pch = 19, cex = 0.4, ylab = "coefficients", res = 600
)
```

```
## Warning in plot.window(...): "res" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "res" is not a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "res" is not a
## graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "res" is not a
## graphical parameter

## Warning in box(...): "res" is not a graphical parameter

## Warning in title(...): "res" is not a graphical parameter
```

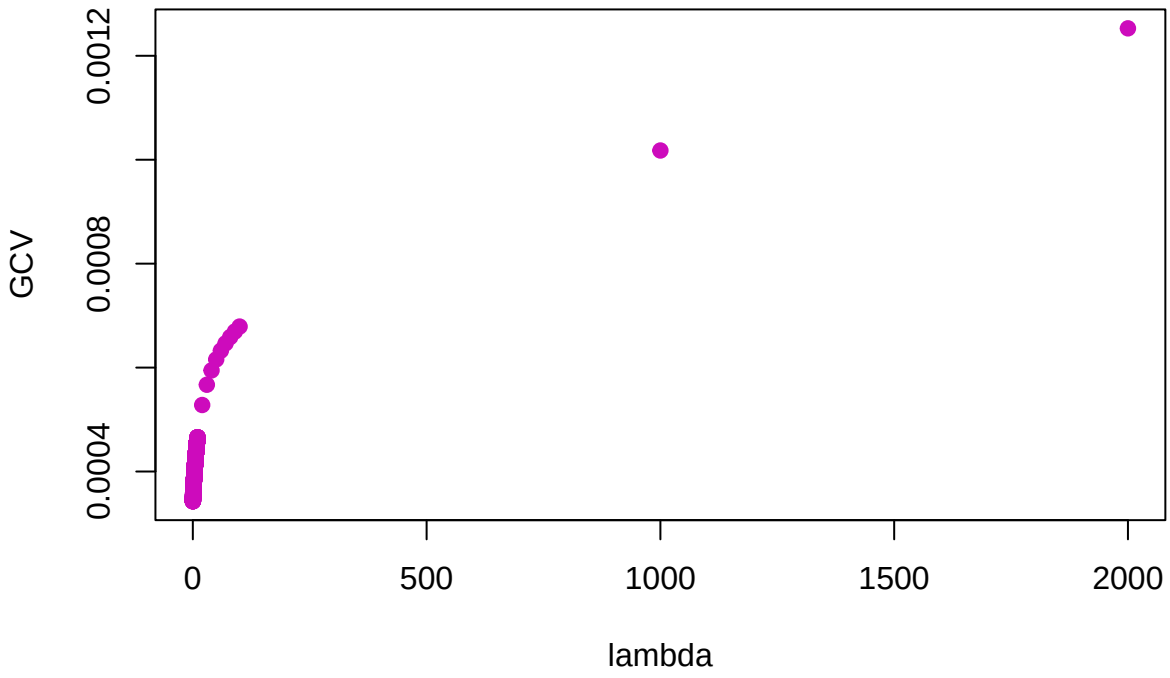
```r
lines(lambda, beta_mat[1, ], lwd = 2)

for (i in 2:(p-1))
{
  points(lambda, beta_mat[i, ], col = i, pch = 19, cex=0.4)
  lines(lambda, beta_mat[i, ], col = i, lwd = 2)

}

legend(1000, 2.5, row.names(beta_mat),
       col = 1:p, lty = rep(1, p), cex = 0.5, pch =  rep(19, 8)
)
```



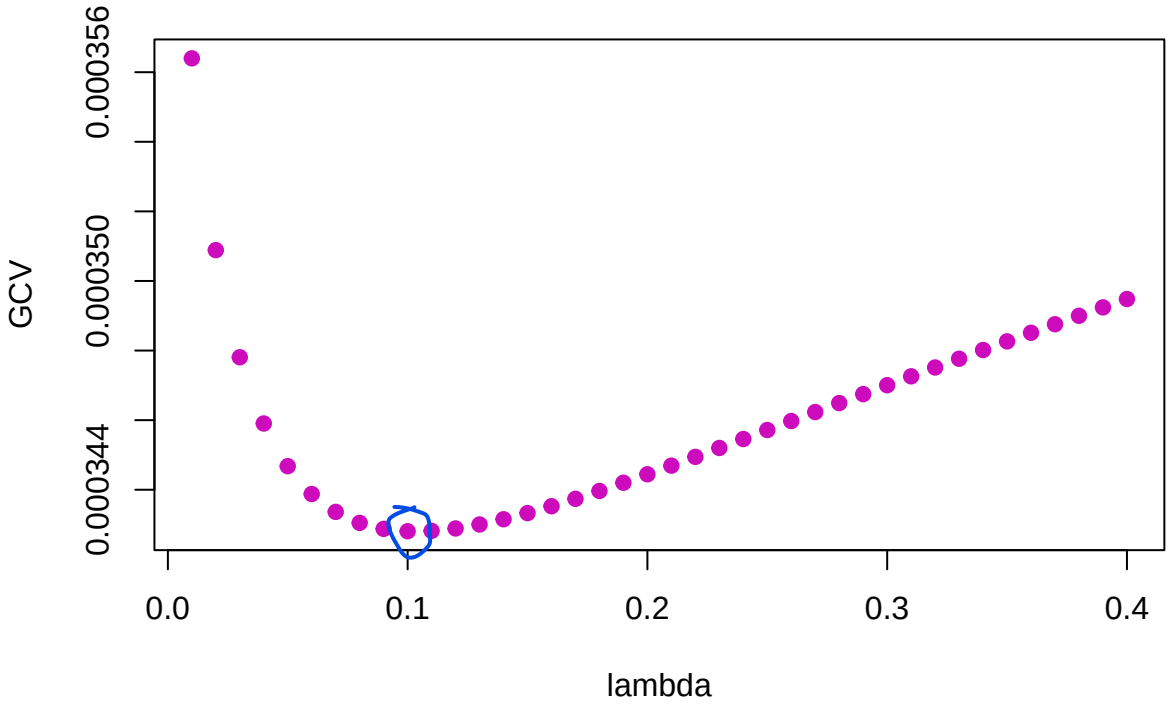
Nesse caso os termos não vão diretamente à zero, por mais que se aproximem dele.

Le choix du meilleur λ se fait par validation croisée.

```r
# tracer du GCV en fonction du lambda
plot(lambda, mod_ridge$GCV, pch = 19, col = 6, xlab = "lambda", ylab = "GCV")
```




```
plot(lambda[1:40], mod_ridge$GCV[1:40], pch = 19, col = 6, xlab = 'lambda', ylab = "GCV")
```




```
# choix du meilleur beta. On récupère la position (indice) où le GCV est minimum
indice = 10  # 0.1 : command which.min()    match(min(mod_ridge$GCV),mod_ridge$GCV)
beta = beta_mat[, indice]
```

La remise à l'échelle est nécessaire pour la prédiction.

```
#prédiction du modèle et remise à l'échelle pour comparer à diab.test
ypred =  as.matrix((x_extend-t(matrix(rep(mean.train[-p2],300),p2-1,300)))/t(matrix(rep(sd.train[-p2],3

X_test = as.matrix(newx - t(matrix(rep(mean.train[-p2], 206), p2-1, 206)))
X_test = X_test / t(matrix(rep(sd.train[-p2], 206), p2-1, 206))
Y_pred_ridge = (as.matrix(X_test)%*%matrix(beta,p2-1,1))*sd.train[p2]+mean.train[p2]

RMSE_ridge = RMSE(housing.test[, p], Y_pred_ridge)
c(RMSE_lm_inter, RMSE_lasso, RMSE_ridge)
```

```
## [1] 4.416989 4.581404 4.653380
```

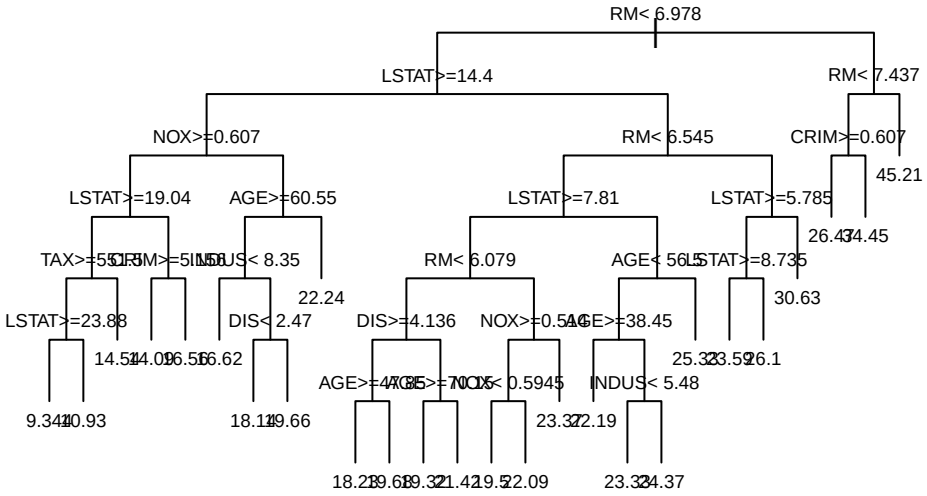
Modèles à base d'arbres

On commence par ajuster un modèle CART de la librairie Rpart.

Classification and regression tree

```
#----------------------------------------
#ajustement cart + random forest
#----------------------------------------
library(rpart)
cont = rpart.control(cp = 0.0001)  # define minimum cp
mod_tree <- rpart(class ~., data = housing.train, control = cont)
par(mfrow = c(1, 1))
plot(mod_tree, uniform = TRUE, margin = 0.05)
text(mod_tree, cex = 0.6)
```

complexity parameter



- A quoi correspond le paramètre *cp* ? Il contrôle la complexité de l'arbre (complexity parameter).
- Le choix de la valeur 0.0001 conduit-il à un arbre grossier ou détaillé ? Un *cp* petit entraine un arbre profond avec beaucoup de noeuds.
- Quelles sont les variables les plus influentes ? RM et LSTAT (look both the height and frequency)
- Pourquoi n'est-il pas nécessaire ici de mettre en place un modèle avec interaction ? Trees already consider the interactions effects
- Faire un choix du meilleur *cp* par validation croisée (quatrième colonne de l'attribut *cptable* d'un objet de type *rpart*).

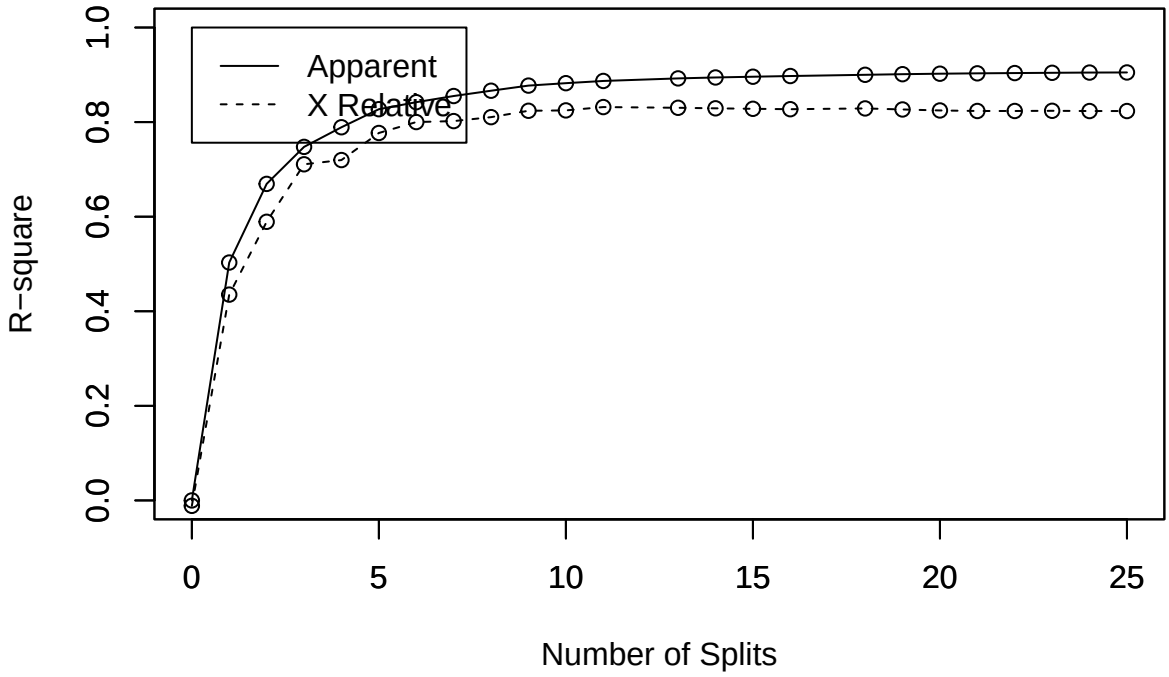
```
#Apparent : apprentissage, X relative : par validation croisée
mod_tree$cptable # ele testa cps maiores (árvores mais simples), antes de chegar em cp=0.0001
```

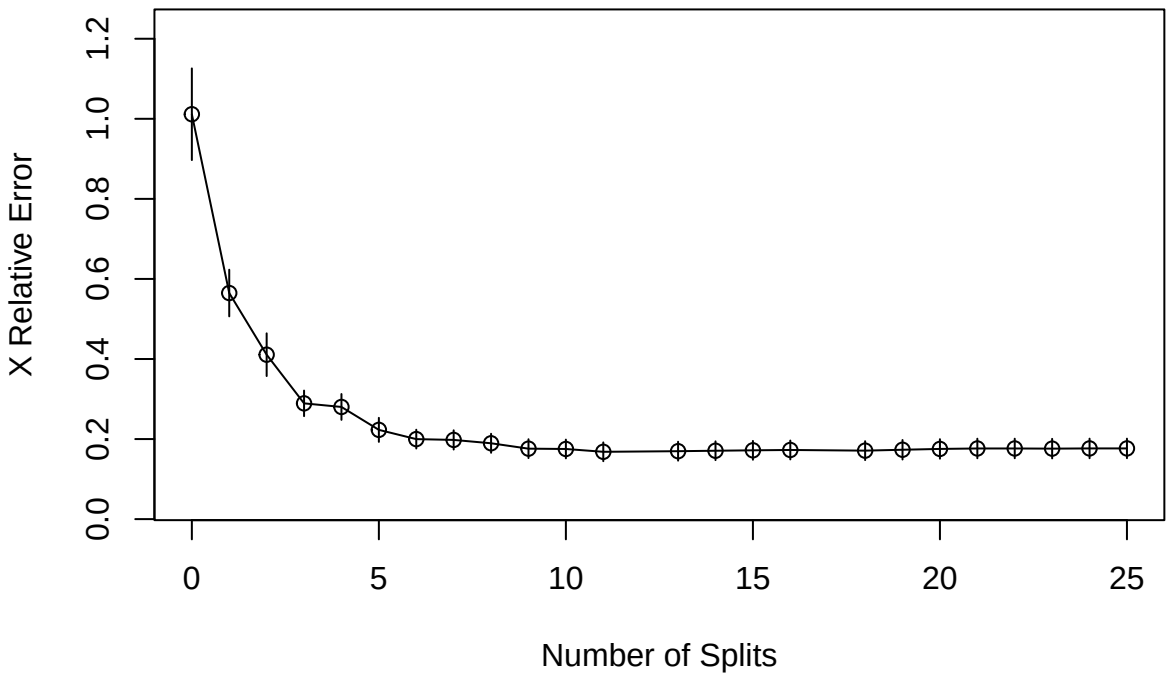
```
##              CP nsplit  rel error    xerror       xstd
## 1  0.5030656136      0 1.00000000 1.0114478 0.11451713
## 2  0.1662686916      1 0.49693439 0.5647737 0.05827073
## 3  0.0782727816      2 0.33066569 0.4107676 0.05333136
## 4  0.0414868184      3 0.25239291 0.2892056 0.03197709
## 5  0.0382249961      4 0.21090609 0.2802356 0.03224474
## 6  0.0148019951      5 0.17268110 0.2229881 0.03005734
## 7  0.0130550489      6 0.15787910 0.1998693 0.02349025
## 8  0.0111708195      7 0.14482405 0.1978706 0.02406715
## 9  0.0108478201      8 0.13365324 0.1894953 0.02378675
## 10 0.0051137140      9 0.12280542 0.1760483 0.02352917
## 11 0.0047095058     10 0.11769170 0.1752187 0.02347684
## 12 0.0027083741     11 0.11298220 0.1681780 0.02349925
## 13 0.0019607608     13 0.10756545 0.1697190 0.02364828
## 14 0.0015930841     14 0.10560469 0.1708474 0.02350104
## 15 0.0014471054     15 0.10401160 0.1718676 0.02353299
## 16 0.0013119307     16 0.10256450 0.1727270 0.02363719
## 17 0.0012952061     18 0.09994064 0.1710708 0.02362531
## 18 0.0009682042     19 0.09864543 0.1732953 0.02440385
## 19 0.0008134445     20 0.09767723 0.1753319 0.02442750
## 20 0.0005885364     21 0.09686378 0.1764818 0.02444670
## 21 0.0005644762     22 0.09627524 0.1764812 0.02444766
## 22 0.0004751045     23 0.09571077 0.1761164 0.02445440
## 23 0.0002569774     24 0.09523566 0.1766070 0.02447482
## 24 0.0001000000     25 0.09497869 0.1766223 0.02447450
```

```
#affichage uniquement cp>0.01
#summary(mod_tree, cp=0.01)

par(mfrow = c(1, 1)) # one plot on one page
rsq.rpart(mod_tree) # visualize cross-validation results
```

```
##
## Regression tree:
## rpart(formula = class ~ ., data = housing.train, control = cont)
##
## Variables actually used in tree construction:
## [1] AGE   CRIM  DIS   INDUS LSTAT NOX   RM    TAX
##
## Root node error: 22208/300 = 74.025
##
## n= 300
##
##            CP nsplit rel error  xerror     xstd
## 1  0.50306561      0  1.000000 1.01145 0.114517
## 2  0.16626869      1  0.496934 0.56477 0.058271
## 3  0.07827278      2  0.330666 0.41077 0.053331
## 4  0.04148682      3  0.252393 0.28921 0.031977
## 5  0.03822500      4  0.210906 0.28024 0.032245
## 6  0.01480200      5  0.172681 0.22299 0.030057
## 7  0.01305505      6  0.157879 0.19987 0.023490
## 8  0.01117082      7  0.144824 0.19787 0.024067
## 9  0.01084782      8  0.133653 0.18950 0.023787
## 10 0.00511371      9  0.122805 0.17605 0.023529
```

```
## 11 0.00470951     10  0.117692 0.17522 0.023477
## 12 0.00270837     11  0.112982 0.16818 0.023499
## 13 0.00196076     13  0.107565 0.16972 0.023648
## 14 0.00159308     14  0.105605 0.17085 0.023501
## 15 0.00144711     15  0.104012 0.17187 0.023533
## 16 0.00131193     16  0.102564 0.17273 0.023637
## 17 0.00129521     18  0.099941 0.17107 0.023625
## 18 0.00096820     19  0.098645 0.17330 0.024404
## 19 0.00081344     20  0.097677 0.17533 0.024427
## 20 0.00058854     21  0.096864 0.17648 0.024447
## 21 0.00056448     22  0.096275 0.17648 0.024448
## 22 0.00047510     23  0.095711 0.17612 0.024454
## 23 0.00025698     24  0.095236 0.17661 0.024475
## 24 0.00010000     25  0.094979 0.17662 0.024474
```



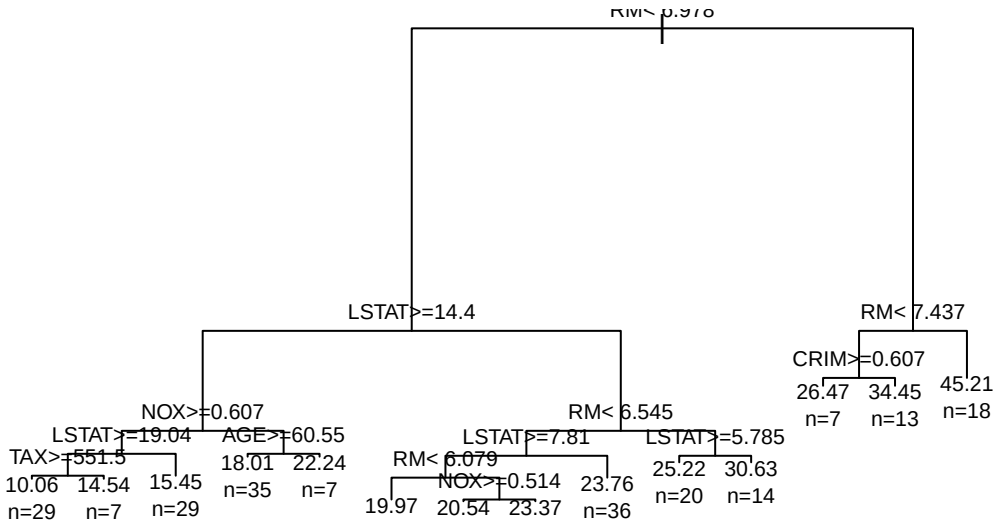



```
# the best result is given by nsplit = 11
mod_tree$cptable[12, 1]
```

```
## [1] 0.002708374
```

- Elaguer l'arbre avec ce choix de *cp* (fonction *prune*)

```
#a completer :
mod_tree_pruned = prune(mod_tree, cp = 0.002708374)
par(mfrow = c(1, 1))
plot(mod_tree_pruned)
text(mod_tree_pruned, use.n = TRUE, cex = 0.7)
```




```
#a completer :
y_pred_tree <- predict(mod_tree_pruned, housing.test[, -p])
#a completer :
RMSE_cart = RMSE(housing.test[, p], y_pred_tree)
RMSE_cart
```

```
## [1] 5.479321
```

- Mettre en place un modèle de boosting en utilisant la routine *gbm* de la librairie *gbm*.
- Combien d'arbres doit-on considérer ? 726
- Quelles sont les variables les plus influentes avec ce modèle ? RM et LSTAT

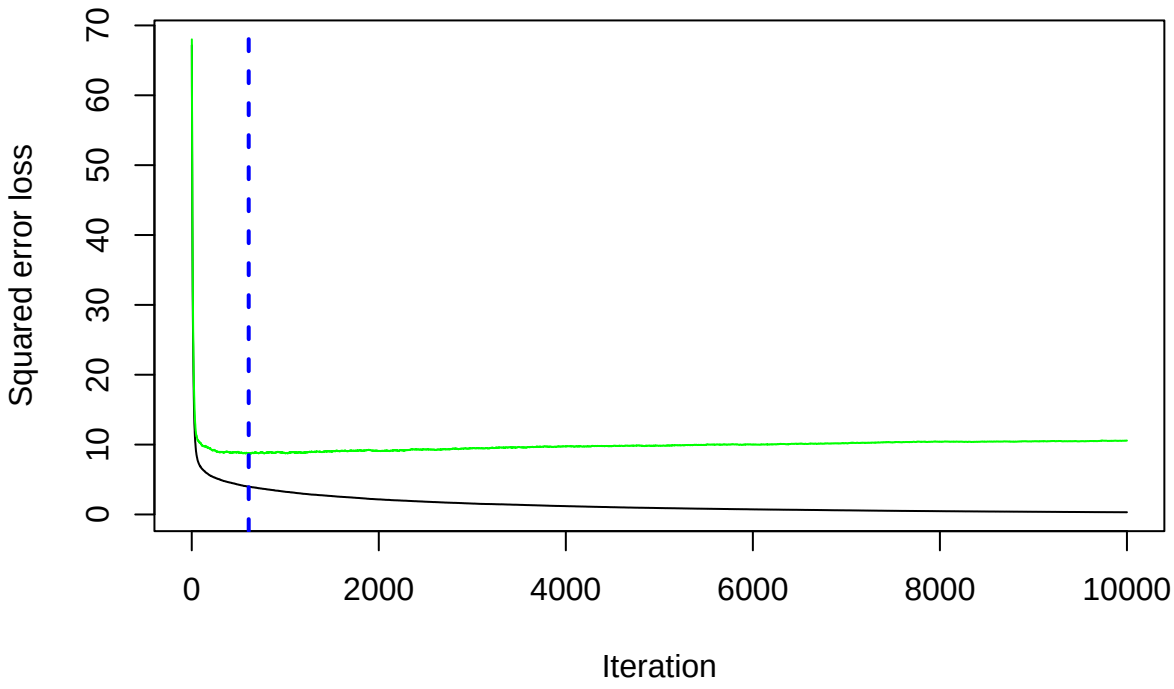
```
#modèle boosted trees
library(gbm)
```

```
## Loaded gbm 2.1.8.1
```

```
mod_boosted <- gbm(class ~., data = housing.train, n.trees = 10000, cv.folds = 10)
```

```
## Distribution not specified, assuming gaussian ...
```

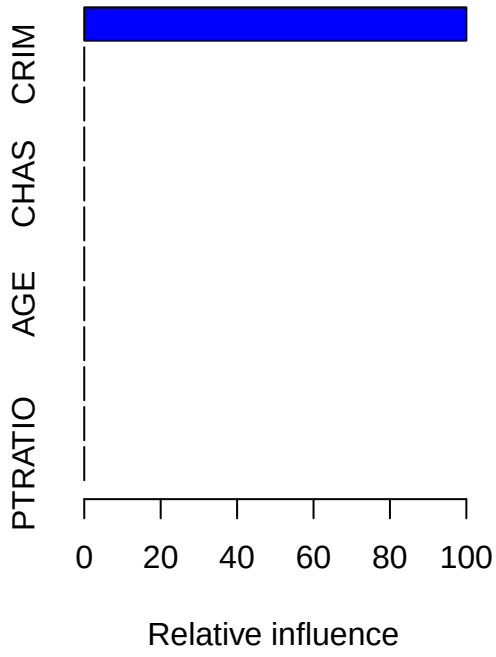
```
par(mfrow = c(1, 1))
best.iter <- gbm.perf(mod_boosted, method = "cv")
```

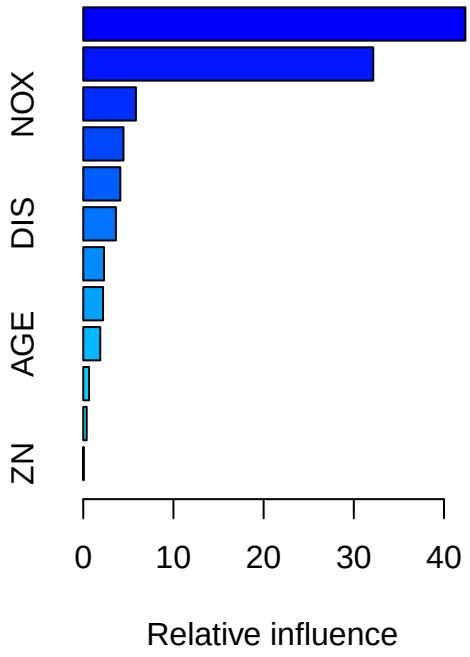



```
par(mfrow = c(1, 2))
summary(mod_boosted, n.trees = 1)          # based on the first tree
```

```
##             var rel.inf
## LSTAT     LSTAT     100
## CRIM       CRIM       0
## ZN           ZN       0
## INDUS     INDUS       0
## CHAS       CHAS       0
## NOX         NOX       0
## RM           RM       0
## AGE         AGE       0
## DIS         DIS       0
## RAD         RAD       0
## TAX         TAX       0
## PTRATIO PTRATIO       0
```

```
summary(mod_boosted, n.trees = best.iter)  # based on the estimated best number of trees
```






```
##             var     rel.inf
## RM           RM 42.36820423
## LSTAT     LSTAT 32.15865779
## NOX         NOX  5.83833014
## CRIM       CRIM  4.46132510
## PTRATIO PTRATIO  4.10908279
## DIS         DIS  3.61888325
## TAX         TAX  2.30171792
## INDUS     INDUS  2.19312476
## AGE         AGE  1.88764093
## RAD         RAD  0.64305519
## CHAS       CHAS  0.38004279
## ZN           ZN  0.03993511
```

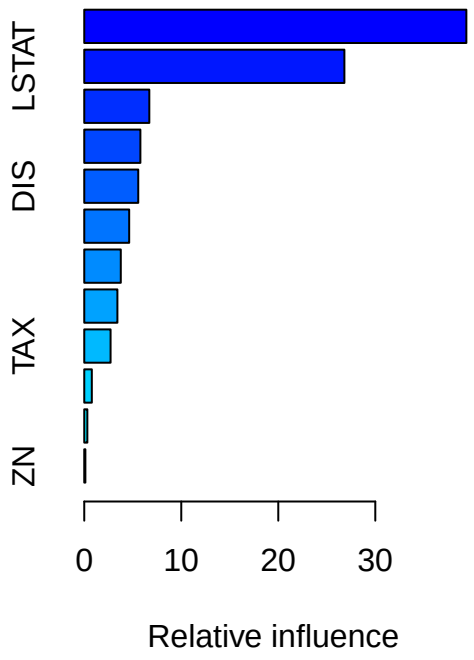
```
summary(mod_boosted, n.trees = 3000)       # based on something
```

```
##             var    rel.inf
## RM           RM 39.3864807
## LSTAT     LSTAT 26.8133470
## CRIM       CRIM  6.7066555
## NOX         NOX  5.7859931
## DIS         DIS  5.5650171
## PTRATIO PTRATIO  4.6317920
## AGE         AGE  3.7671004
## INDUS     INDUS  3.4174186
```

```
## TAX         TAX  2.7119535
## RAD         RAD  0.7809762
## CHAS       CHAS  0.3185178
## ZN           ZN  0.1147481
```

```r
#print(pretty.gbm.tree(mod_boosted,1))
#print(pretty.gbm.tree(mod_boosted,mod_boosted$n.trees))
f.predict1 <- predict(mod_boosted, housing.test[, -p], 1)
f.predict2 <- predict(mod_boosted, housing.test[, -p], 3000)
f.predict3 <- predict(mod_boosted, housing.test[, -p], best.iter)
RMSE_b1 =  RMSE(housing.test[, p], f.predict1)
RMSE_b2 =  RMSE(housing.test[, p], f.predict2)
RMSE_b3 =  RMSE(housing.test[, p], f.predict3)
c(RMSE_b1, RMSE_b2, RMSE_b3)
```

```
## [1] 9.618313 5.024867 4.853227
```



bagging = selecionar aleatoriamente conjunto de linhas dos dados de treino, para treinar modelos em sequencia
boosting = selecionar priorizando dados que foram pior preditos, para treinar modelos em sequencia

On construit maintenant une procédure de bagging. L'idée est de moyenner des arbres CART construits sur des échantillons bootstrap des données de départ. Un échantillon Bootstrap est un tirage avec remise de N points parmi les N points de l'échantillon initial.

```r
#modèle arbre bagging
cont = rpart.control(minsplit = 2, cp = 0.0001)
B = 2000
Y <- matrix(0, nrow(housing.test), B)
```

```
for (i in 1:B) {
  u = sample(1:nrow(housing.train), nrow(housing.train), replace = TRUE)
  appren <- housing.train[u, ]
  mod_tree <- rpart(class ~., data = appren, control = cont)
  Y[, i] <- predict(mod_tree, newdata = housing.test)
}

#Y est de taille 206*B, 206 etant le nombre de points tests
#a completer :
Y_pred_bag = apply(Y, 1, mean)  # mean of each tree`s result
#a completer :
RMSE_bag = RMSE(housing.test[, p], Y_pred_bag)
RMSE_bag
```

Bagging is an ensemble algorithm that fits multiple models on different subsets of a training dataset, then combines the predictions from all models.

Random forest is an extension of bagging that also randomly selects subsets of features used in each data sample.

```
## [1] 4.864175
```

Construire un modèle de Type Random Forest. - quelle est la différence entre un modèle type bagging et un modèle de forets aléatoires ?

```
#modèle arbres random forest
library(randomForest)
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
mod_RF <- randomForest(class~., data = housing.train, ntree = 300, sampsize = nrow(housing.train))

summary(mod_RF)
```

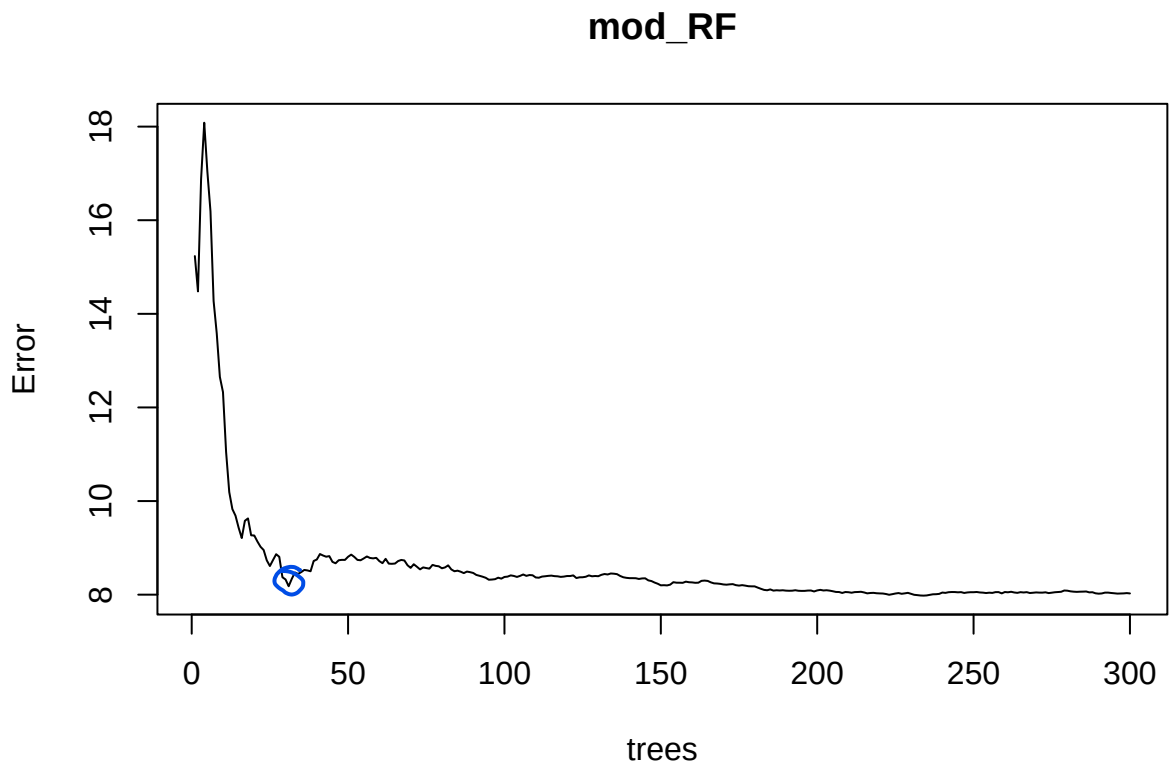
```
##                 Length Class  Mode
## call              5    -none- call
## type              1    -none- character
## predicted       300    -none- numeric
## mse             300    -none- numeric
## rsq             300    -none- numeric
## oob.times       300    -none- numeric
## importance       12    -none- numeric
## importanceSD      0    -none- NULL
## localImportance   0    -none- NULL
## proximity         0    -none- NULL
## ntree             1    -none- numeric
## mtry              1    -none- numeric
```

```
## forest           11    -none- list
## coefs             0    -none- NULL
## y               300    -none- numeric
## test              0    -none- NULL
## inbag             0    -none- NULL
## terms             3    terms  call
```

```
importance(mod_RF)
```

```
##         IncNodePurity
## CRIM       1289.00260
## ZN          194.06135
## INDUS      1409.38286
## CHAS         41.82852
## NOX        1309.59676
## RM         7471.05953
## AGE         656.74447
## DIS        1036.39422
## RAD         144.51651
## TAX        1059.09902
## PTRATIO    1306.87398
## LSTAT      5852.45326
```

```
plot(mod_RF)
```



errado, precisava ter pegado o minimo do erro

```
which(mod_RF$trees == min(mod_RF$trees), arr.ind = TRUE)
```

```
## Warning in min(mod_RF$trees): no non-missing arguments to min; returning Inf
```

```
## integer(0)
```

talvez esteja errado porque precisava especificar qual árvore usar

```
#a completer :
Y_pred_RF <- predict(mod_RF,housing.test)
#a completer :
RMSE_RF = RMSE(housing.test[, p], Y_pred_RF)
RMSE_RF
```

```
## [1] 4.931825
```

Méthode de réduction de dimension par orthogonalisation

On s'intéresse maintenant aux modèles PCR et PLS construits à partir de la matrice augmentée des interactions.

```
library(pls)
```

```
##
## Attaching package: 'pls'
```

```
## The following object is masked from 'package:caret':
##
##     R2
```

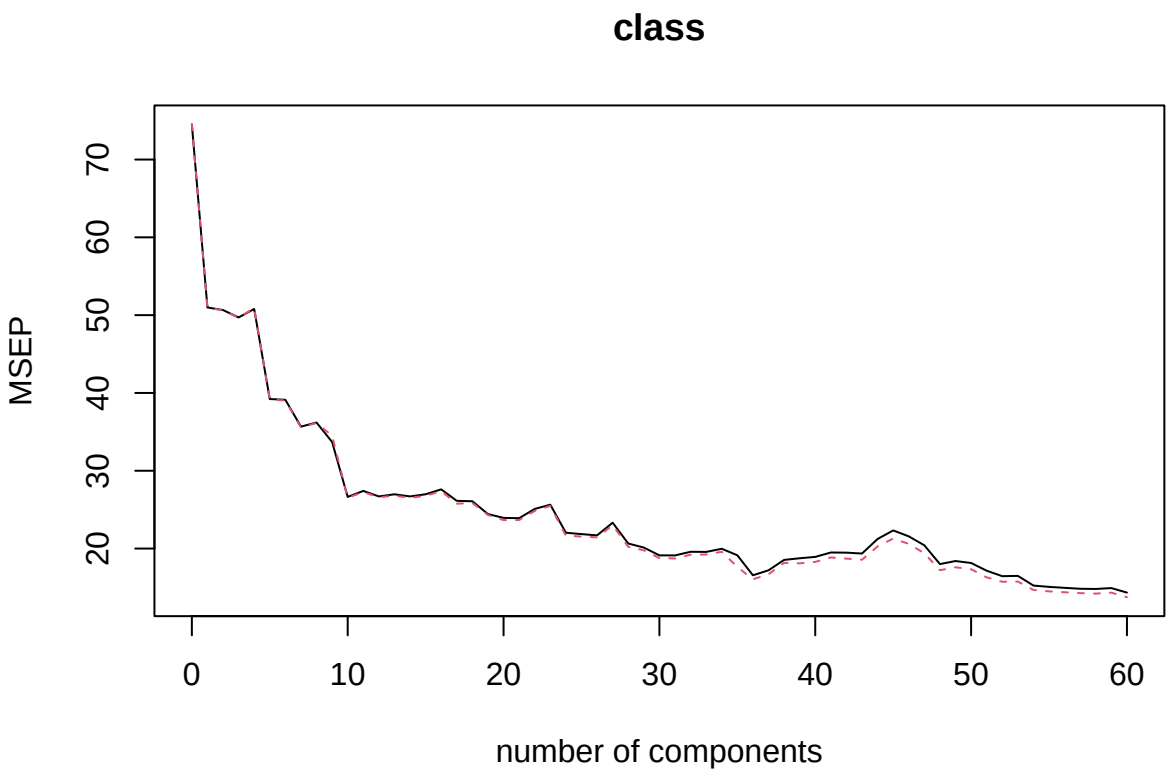
```
## The following object is masked from 'package:DiceEval':
##
##     R2
```

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

```
#PCR
housing.pcr <- pcr(class ~ .^2, 60, data = housing.train)
```

Faire le choix du nombre de composantes par validation croisée en utilisant la routine crossval.

```
#a completer :
housing.cv <- crossval(housing.pcr,segments = 10)
plot(MSEP(housing.cv))
```



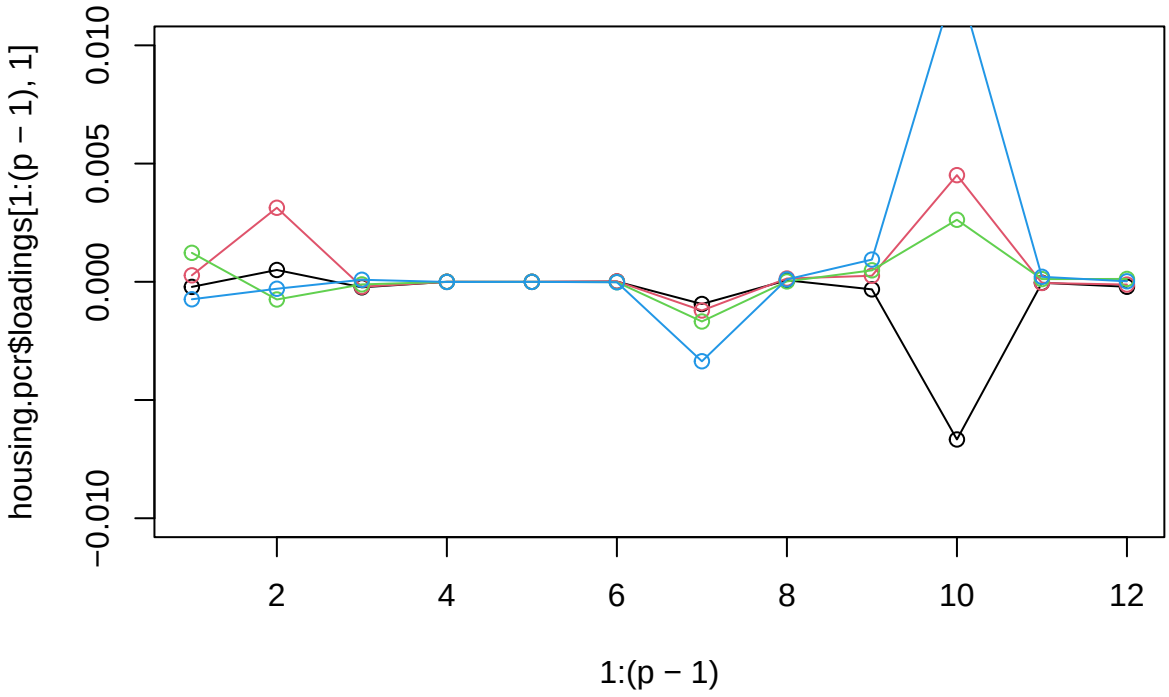
Faire la prédiction du modèle pour le nombre de composantes sélectionnées.

```
#a completer :
nbcomp = 36
Y_PCR = predict(housing.pcr, newdata = housing.test, ncomp = nbcomp, type = "response")
#a completer :
RMSE_PCR = RMSE(housing.test[, p], Y_PCR)
RMSE_PCR
```

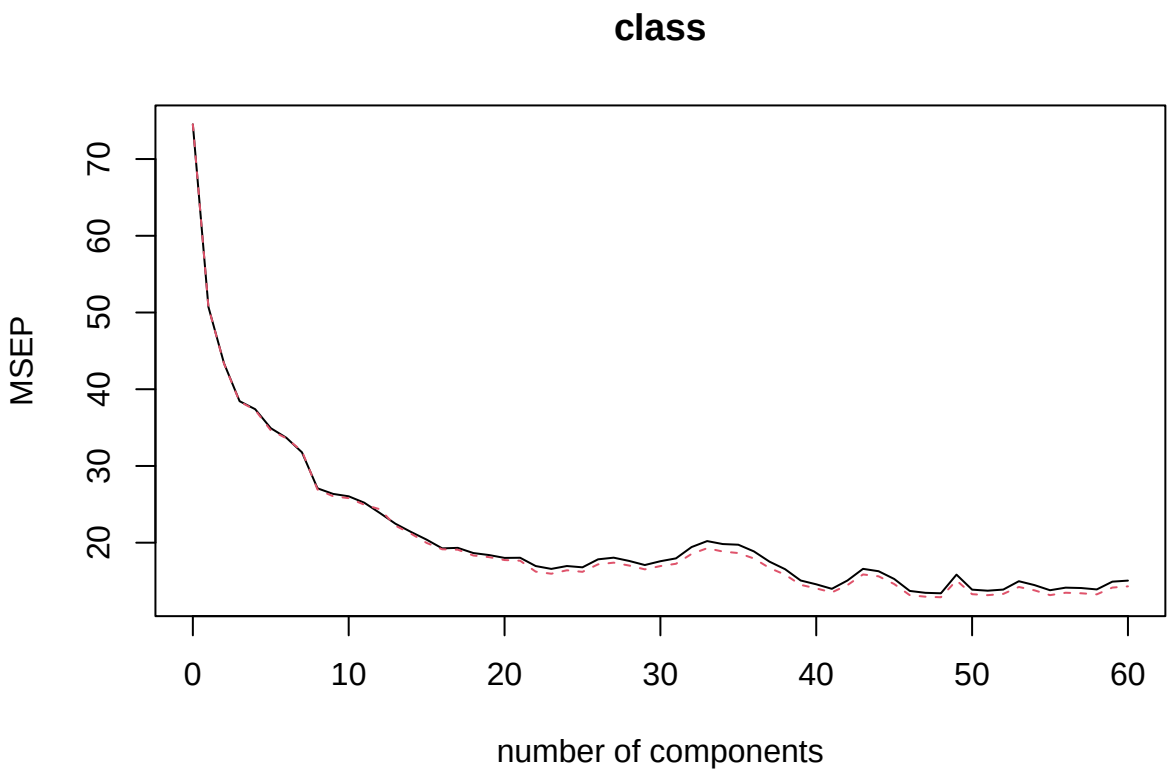
```
## [1] 5.487672
```

Tracer les premières fonctions propres et interpréter.

```
housing.pcr <- pcr(class ~ .^2, nbcomp, data = housing.train)
#coef(housing.pcr)

#tracer des premières fonctions propres
par(mfrow =c(1,1))
plot(1:(p-1),housing.pcr$loadings[1:(p-1),1],type ="l",col = 1,ylim =c(-0.01,0.01))
points(1:(p-1),housing.pcr$loadings[1:(p-1),1],col = 1)
for (i in 2:4)
{
  points(1:(p-1),housing.pcr$loadings[1:(p-1),i],col = i)
  lines(1:(p-1),housing.pcr$loadings[1:(p-1),i],col = i)
}
```




```
#PLS
housing.pls <- plsr(class ~ .^2, 60, data = housing.train)
housing.pls.cv <- crossval(housing.pls,segments =10)
plot(MSEP(housing.pls.cv))
```




```
#a completer :
Y_PLS = predict(housing.pls, newdata = housing.test, ncomp = nbcomp, type = "response")
#a completer :
RMSE_PLS = RMSE(housing.test[, p], Y_PLS)
RMSE_PLS
```

```
## [1] 4.993442
```