Linear models

(attach(data): $X_1, .., X_p, Y$)

```
###-------------------------------------------------------------------------
###-------------------------------------------------------------------------
### Linear models
###-------------------------------------------------------------------------
###-------------------------------------------------------------------------
library(MASS)
library(car)
library(rgl)

# ------------------------------------------------------------------------------
# ------------------------------------------------------------------------------
# PART 1 - Linear models, plot(p=1/2), clustering
# ------------------------------------------------------------------------------
# ------------------------------------------------------------------------------

# ------------------------------------------------------------------------------
# Setting
# (Y colonna finale)
# ------------------------------------------------------------------------------
data = cars
data[, 'speed^2'] = cars$speed^2
data = data[,-2]
data[, 'dist']    = cars$dist
head(data)

# Rinominiamo
n = dim(data)[1]
p = dim(data)[2]-1
v = 'X1'
if(p>1){
  for(i in 2:p){
    v = c(v, paste('X',i,sep=''))
  }
}

v = c(v, 'Y')
v
colnames(data) = v
head(data)
attach(data)

# ------------------------------------------------------------------------------
# Exploring
# (da modificare se p!=2)
# ------------------------------------------------------------------------------
plot(X1, Y, xlab='X1', ylab='Y')
plot(X2, Y, xlab='X2', ylab='Y')

pairs(data)
plot3d(X1, X2, Y, size=4, asp = T)

# ------------------------------------------------------------------------------
# MODEL
# ------------------------------------------------------------------------------
#   Y = beta_0 + beta_1 * X1 + beta_2 * X2 + Eps
## Assumptions:
#    1) Parameter estimation: E(Eps) = 0  and  Var(Eps) = sigma^2
#    2) Inference           : Eps ~ N(0, sigma^2)

# Estimate the parameters
fm = lm(Y ~ X1 + X2 )
summary(fm)
vif(fm)

# Comment
#  * Residual standard error = estimate of sigma
#  * Degrees of freedom      = n-(r+1)
#  * F-statistic - p.value   = H0: beta_1 = .. = beta_r = 0
#                              p-value basso => ha senso fare il testo
#  * CONTROLLA R^2 e R.adj^2
#  * i residui devono essere simmetrici bene o male
#  * Pr(>|t|) = p-value dei test one-at-time dei beta, attenzione, togliere 1 alla volta

# y hat
fitted(fm)                  # ŷ

# eps hat
residuals(fm)               # ε̂
plot(residuals(fm))

# beta hat
coefficients(fm)            # β̂

# cov(beta hat)  === sigma^2 * (Z^T * Z)^(-1)
vcov(fm)                    # σ²(ZᵀZ)⁻¹
```

```r
# order of the model (r+1)
fm$rank

# degrees of freedom for the residuals
fm$df

# leverages  h_ii
hatvalues(fm)

# standardized residuals
rstandard(fm)
plot(rstandard(fm))

# estimate of sigma^2
sum(residuals(fm)^2)/fm$df

# ------------------------------------------------------------------------------
# Plot della regressione
# (caso specifico di una regressione che era: Y = b_0 + b_1 * X + b_2 * X^2)
# ------------------------------------------------------------------------------
x = seq(0,30,by=0.01)
b = coef(fm)
plot(X1, Y, xlab='X1', ylab='Y')
lines(x, b[1]+b[2]*x+b[3]*x^2)

### --------------------------------------
# Se p = 2
par3d(windowRect=c(680,40,1350,720))
points3d(x=X1, y=X2, z=Y, size=4, aspect = T)
box3d()
axes3d()
points3d(x=X1, y=X2, z=fitted(fm), size=4, col = 'blue')
surface3d(range(data$X1), range(data$X2),
          matrix(predict(fm, expand.grid(X1=range(X1), X2=range(X2))),2,2),alpha = 0.5)
### --------------------------------------

# ------------------------------------------------------------------------------
# INFERENCE ON BETA's
# ------------------------------------------------------------------------------
## Assuption: Eps ~ N(0, sigma^2)
## Test (Fisher):
## H0: (beta1, beta2) == (0, 0)
## H1: (beta1, beta2) != (0, 0)

linearHypothesis(fm, rbind(c(0,1,0), c(0,0,1)), c(0,0))
# con più beta (per esempio 3):
# linearHypothesis(fm, rbind(c(0,1,0,0), c(0,0,1,0), c(0,0,0,1)), c(0,0,0))

# Comment
# Pr(>F) = p-value finale in summary(fm)

r = fm$rank - 1

### --------------------------------------
### CONFIDENCE REGION (p=2)
# Center
c(coefficients(fm)[2], coefficients(fm)[3])

# Direction of the axes
eigen(vcov(fm)[2:3, 2:3])$vectors

plot(coefficients(fm)[2], coefficients(fm)[3], xlim = c(-6,6), ylim = c(-6,6),
     asp=1, xlab='beta1', ylab='beta2')
ellipse(coefficients(fm)[2:3], vcov(fm)[2:3,2:3], sqrt(p*qf(1-0.05,p,n-(r+1))))
abline(v=0)
abline(h=0)

# Comment
# Se è squishata allora probabilmente sono collineari
### --------------------------------------

### --------------------------------------
### BONFERRONI INTERVALS (level 95%) (p=2)
Bf <- rbind(
  beta1=c(coefficients(fm)[2]-sqrt(vcov(fm)[2,2])*qt(1-0.05/(2*p), n-(r+1)),
          coefficients(fm)[2]+sqrt(vcov(fm)[2,2])*qt(1-0.05/(2*p), n-(r+1))),
  beta2=c(coefficients(fm)[3]-sqrt(vcov(fm)[3,3])*qt(1-0.05/(2*p), n-(r+1)),
          coefficients(fm)[3]+sqrt(vcov(fm)[3,3])*qt(1-0.05/(2*p), n-(r+1)))
)
Bf

# Generico beta_j  (p, r, n, alpha generici)
#     beta_j = c(coefficients(fm)[j]-sqrt(vcov(fm)[j,j])*qt(1-alpha/(2*p), n-(r+1)),
#                coefficients(fm)[j]+sqrt(vcov(fm)[j,j])*qt(1-alpha/(2*p), n-(r+1)))

# ALTERNATIVAMENTE: Bonferroni's correction
confint(fm, level= 1-0.05/p)[2:3,]
```

Inference on Bs

```
### ----------------------------------------

### ----------------------------------------
### ALTRI TEST CON "linearHypothesis"
# Test:
# H0: (beta0+beta2, beta1) == (0,0)
# H1: (beta0+beta2, beta1) != (0,0)

C <- rbind(c(1,0,1), c(0,1,0))
linearHypothesis(fm, C, c(0,0))

# INTERVALLI DI CONFIDENZA SIMULTANEI combinazioni di beta_j
#    quantile Fisher 1-alpha
alpha    = 0.05
qf.fish = qf(1-alpha, r+1, n-(r+1))

#    combinazione scelta (qui prendo beta_1)
a = c(0,1,0)

#    intervallo
sim_IC = c(t(a)%*%coefficients(fm) - sqrt(t(a)%*%vcov(fm)%*%a) * sqrt((r+1)*qf.fish),
           t(a)%*%coefficients(fm) + sqrt(t(a)%*%vcov(fm)%*%a) * sqrt((r+1)*qf.fish))
sim_IC
### ----------------------------------------

# ------------------------------------------------------------------------------------------
# INFERENCE ON THE MEAN
#  CI(E[Y|X]) and PI(Y)
# ------------------------------------------------------------------------------------------
# Nuovo dato
Z0.new = data.frame(X1=10, X2=10^2)

alpha = 0.05

# CI(E[Y|X])
Conf = predict(fm, Z0.new, interval='confidence', level=1-alpha)
Conf

# PI(Y)
Pred = predict(fm, Z0.new, interval='prediction', level=1-alpha)
Pred

### ----------------------------------------
# Plot IC vs IP: un dato
# (particolare per il modello dell'esercizio)
plot(X1, Y, xlab='X1', ylab='Y', las=1)
x = seq(0,30,by=0.1)
b = coef(fm)
lines(x, b[1]+b[2]*x+b[3]*x^2)
points(10,Conf[1], pch=19)
segments(10,Pred[2],10,Pred[3],col='gold', lwd=2)
segments(10,Conf[2],10,Conf[3],col='red', lwd=2)
points(10,Conf[2], pch='-', col='red', lwd=2)
points(10,Conf[3], pch='-', col='red', lwd=2)
points(10,Pred[2], pch='-', col='gold', lwd=2)
points(10,Pred[3], pch='-', col='gold', lwd=2)
### ----------------------------------------

### ----------------------------------------
# Plot IC vs IP: griglia di dati
Z0   <- data.frame(cbind(X1 = seq(min(X1), max(X1), length=100),
                         X2 = seq(min(X2), max(X2), length=100)))
Conf <- predict(fm, Z0, interval='confidence')
Pred <- predict(fm, Z0, interval='prediction')

plot(X1, Y, xlab='X1', ylab='Y', las=1)
lines(Z0[,1], Conf[,'fit'])
lines(Z0[,1], Conf[,'lwr'], lty=2, col='red', lwd=2)
lines(Z0[,1], Conf[,'upr'], lty=2, col='red', lwd=2)
lines(Z0[,1], Pred[,'lwr'], lty=3, col='gold', lwd=2)
lines(Z0[,1], Pred[,'upr'], lty=3, col='gold', lwd=2)

# Comment
# NON sono confidence/prendiction bands, perché sono fatti uno alla volta
### ----------------------------------------

# ------------------------------------------------------------------------------------------
# VERIFY ASSUMPTIONS (used for inference and estimate of parameters)
#    * gaussianity
#    * homoschedasticity
# ------------------------------------------------------------------------------------------
par(mfrow=c(2,2))
plot(fm)
dev.off()

# Comment
#  1. we want to see no pattern: a cloud around the zero
```

Inference on Bs

Inference on E[Y|X], Y

Inf. on E[Y|X], Y

Inf. on Bs

```
#  2. We want to see a good fit on the line
#  3. again, we want to see no pattern
#  4. we have the iso-lines of the Cook distance: we can identify the outliers

shapiro.test(residuals(fm))

detach(data)

# ------------------------------------------------------------------------------
# Plots - fiketti
# ------------------------------------------------------------------------------
plot(X2, Y)
text(X2, Y)

# se fitto con un solo regressore al plot posso aggiungere:
abline(coef(fm)[1], coef(fm)[2])

### ------------------------------------------------------------------------------
### SE TRASFORMO I DATI
### ------------------------------------------------------------------------------
data = read.table('brain_weight.txt', head=T)
head(data)

# Rinominiamo
n = dim(data)[1]
p = dim(data)[2]-1
v = 'X1'
if(p>1){
  for(i in 2:p){
    v = c(v, paste('X',i,sep=''))
  }
}

v = c(v, 'Y')
v
colnames(data) = v
head(data)
attach(data)

# Linear regression
logX1 = log(X1)
logY  = log(Y)
fm = lm(logY~logX1)

# Plot
plot(logX1, logY)
abline(coef(fm)[1], coef(fm)[2])

### ---------------------------------------
# Plot IC vs IP: griglia di dati TRASFORMATI
logZ0 <- data.frame(logX1=seq(min(logX1), max(logX1), length=100))
Conf  <- predict(fm, logZ0, interval='confidence')
Pred  <- predict(fm, logZ0, interval='prediction')

plot(logX1, logY, xlab='logX1', ylab='logY', las=1)
lines(logZ0[,1], Conf[,'fit'])
lines(logZ0[,1], Conf[,'lwr'], lty=2, col='red', lwd=2)
lines(logZ0[,1], Conf[,'upr'], lty=2, col='red', lwd=2)
lines(logZ0[,1], Pred[,'lwr'], lty=3, col='gold', lwd=2)
lines(logZ0[,1], Pred[,'upr'], lty=3, col='gold', lwd=2)

# Comment
# NON sono confidence/prendiction bands, perché sono fatti uno alla volta
### ---------------------------------------

### ---------------------------------------
# Plot IC vs IP: griglia di dati TRASFORMATI
plot(X1, Y, xlab='X1', ylab='Y', las=1)
CI = exp(Conf)
PI = exp(Pred)
Z0 = exp(logZ0)
lines(Z0[,1], CI[,'fit'])
lines(Z0[,1], CI[,'lwr'], lty=2, col='red', lwd=2)
lines(Z0[,1], CI[,'upr'], lty=2, col='red', lwd=2)
lines(Z0[,1], PI[,'lwr'], lty=3, col='gold', lwd=2)
lines(Z0[,1], PI[,'upr'], lty=3, col='gold', lwd=2)

# Comment
# NON sono confidence/prendiction bands, perché sono fatti uno alla volta
### ---------------------------------------

detach(data)

### ------------------------------------------------------------------------------
### CASO DISPERATO
### Il modello è terribile: provo a fare un hierchical clustering e aggiungo
### delle dummy variables per il clustering
```

Hierarchical + linear model

```
### ------------------------------------------------------------------------------
# Cluster method: ward
clusterw = cutree(hclust(dist(data), method='ward.D2'),2)
dummy    = clusterw - 1

# Model
# Y = b_0 + b_1*X1 + b_2*X2 + b_3*dummy + b_4*dummy*X1 + b_5*dummy*X2
fm2 = lm(Y ~ X1 + X2 + dummy + X1:dummy + X2:dummy)
summary(fm2)

par(mfrow=c(2,2))
plot(fm2)

### --------------------------------------
### Se p = 2
par3d(windowRect=c(680,40,1350,720))
points3d(x=X1, y=X2, z=Y, size=4, col=clusterw+1, aspect = T)
points3d(x=X1, y=X2, z=fitted(fm2), size=4, col = 'blue')
surface3d(range(X1), range(X2),
          matrix(predict(fm2, expand.grid(X1=range(X1),X2=range(X2), dummy=c(1,1))),2,2),
          alpha = 0.5, col='green')
surface3d(range(X1), range(X2),
          matrix(predict(fm2, expand.grid(X1=range(X1), X2=range(X2), dummy=c(0,0))),2,2),
          alpha = 0.5, col='red')
box3d()
axes3d()
### --------------------------------------

### --------------------------------------
### Test: sono i due plane necessari?
A = rbind(c(0,0,0,1,0,0), c(0,0,0,0,1,0), c(0,0,0,0,0,1))
b = c(0,0,0)
linearHypothesis(fm2, A, b)

# Comment
# Pr(>F) ci dice se i cluster sono necessari
### --------------------------------------

# ------------------------------------------------------------------------------
# ------------------------------------------------------------------------------
# PART 2 - Collinearity: PCA, Ridge, Lasso
# ------------------------------------------------------------------------------
# ------------------------------------------------------------------------------

# ------------------------------------------------------------------------------
# Setting
# (Y colonna finale)
# ------------------------------------------------------------------------------
data = cars
data[, 'speed^2'] = cars$speed^2
data = data[,-2]
data[, 'dist']    = cars$dist
head(data)

# Rinominiamo
n = dim(data)[1]
p = dim(data)[2]-1
v = 'X1'
if(p>1){
  for(i in 2:p){
    v = c(v, paste('X',i,sep=''))
  }
}

v = c(v, 'Y')
v
colnames(data) = v
head(data)
attach(data)

# ------------------------------------------------------------------------------
# Model
# ------------------------------------------------------------------------------
fm = lm(Y ~ X1 + X2)
summary(fm)

# Check for collinearity
vif(fm)

# Note
# Se è maggiore di 10 è considerato alto

# ------------------------------------------------------------------------------
# PCA Regression        - start
# (p=2)
# ------------------------------------------------------------------------------
# PCA
```

PCA - regression

```
data.pc = princomp(cbind(X1,X2), scores=TRUE)
summary(data.pc)
E       = as.matrix(data.pc$loadings[,])
# Note
# In base alla Cumulative Proportion decidiamo quali e quante var prendere

# Note
# Qui si può piazzare dal file 'pca.R'

# PCA regression
#   Y = a_0 + a_1 * X1.pc + a_2 * X2.pc
X1.pc = data.pc$scores[,1]
X2.pc = data.pc$scores[,2]
fm.pc = lm(Y ~ X1.pc + X2.pc)
summary(fm.pc)

### ---------------------------------------
# Ritorno alle variabili originali
# (tutto automatico tranne la media delle Xi)
### ---------------------------------------
#   Y = b_0 + b_1 * X1 + b_2 * X2
# trasformando la PCA regression:
#   b_0 = a_0
#   b_j = (e_j1 * a_1 + .. + e_jp * a_p )

# creo il vettore degli alphas: [a_1, a_2, .., a_k]
k = length(fm.pc$coefficients)-1
alphas = as.vector(fm.pc$coefficients)
alpha0 = alphas[1]
alphas = alphas[2:(k+1)]
alphas = as.matrix(alphas)     # (k x 1)

# prendo i loadings che mi servono (primi k)
E = as.matrix(E[,1:k])         # (p x k)

# creo il vettore dei beta: [b_1, .., b_p]
betas = as.matrix(0*(1:p))     # (p x 1)

# creo il vettore delle medie: [m1, .., mp]
# (MANUALMENTE)
m    = 0*(1:p)
m[1] = mean(X1)
m[2] = mean(X2)
m    = as.matrix(m)            # (p x 1)

# beta_1, .., beta_p
betas = E %*% alphas

# beta 0
b_0 = alpha0 - t(m) %*% betas

# beta insieme
# b_0, b_1, .., b_p
beta_finali = c(b_0, betas)
beta_finali
### ---------------------------------------
```

✓ [verificato]

```
# ------------------------------------------------------------------------------
# PCA Regression           - end
# ------------------------------------------------------------------------------
```

Ridge - reg.

```
# ------------------------------------------------------------------------------
# Ridge Regression         - start
# ------------------------------------------------------------------------------
lambda = 0.5
fm.ridge = lm.ridge(Y ~ X1 + X2, lambda = lambda)

# coefficienti
coef(fm.ridge)

# y.hat
y.hat.ridge = cbind(rep(1,n), X1, X2) %*% coef(fm.ridge)

# optimal lambda
lambda.c=seq(0,10,0.01)
fm.ridge = lm.ridge(Y ~ X1 + X2, lambda = lambda.c)
select(fm.ridge)

# alternative (optimal lambda)
lambda.c=seq(0,10,0.01)
fm.ridge = lm.ridge(Y ~ X1 + X2, lambda = lambda.c)
lambda.opt = lambda.c[which.min(fm.ridge$GCV)]

# coefficienti della lambda ottimale
coef.ridge = coef(fm.ridge)[which.min(fm.ridge$GCV),]
coef.ridge
```

meglio il metodo 2 per la ridge regr.

Ridge - regression

```
# ------------------------------------------------------------------------------
# Ridge Regression         - end
# ------------------------------------------------------------------------------

# ------------------------------------------------------------------------------
# Ridge Regression (2)     - start
# ------------------------------------------------------------------------------
library(glmnet)
x <- model.matrix(Y ~ X1 + X2)[,-1] # matrix of predictors
y <- Y # vector of response
lambda.grid <- 10^seq(5,-3,length=50)

# Ridge regression
fit.ridge <- glmnet(x,y, lambda = lambda.grid, alpha=0) # alpha=0 -> ridge
plot(fit.ridge,xvar='lambda',label=TRUE, col = rainbow(dim(x)[2]))
legend('topright', dimnames(x)[[2]], col =  rainbow(dim(x)[2]), lty=1, cex=1)

# Let's set lambda via CV
cv.ridge <- cv.glmnet(x,y,alpha=0,nfolds=3,lambda=lambda.grid)

bestlam.ridge <- cv.ridge$lambda.min
bestlam.ridge

plot(cv.ridge)
abline(v=log(bestlam.ridge), lty=1)

coef.ridge <- predict(fit.ridge, s=bestlam.ridge, type = 'coefficients')[1:(p+1),]
coef.ridge

plot(fit.ridge,xvar='lambda',label=TRUE, col = rainbow(dim(x)[2]))
abline(v=log(bestlam.ridge))
# ------------------------------------------------------------------------------
# Ridge Regression (2)     - end
# ------------------------------------------------------------------------------
```

Lasso - regression

```
# ------------------------------------------------------------------------------
# Lasso Regression         - start
# ------------------------------------------------------------------------------
library(glmnet)
x <- model.matrix(Y ~ X1 + X2)[,-1] # matrix of predictors
y <- Y # vector of response
lambda.grid <- 10^seq(5,-3,length=50)

# Lasso regression
fit.lasso <- glmnet(x,y, lambda = lambda.grid, alpha=1) # alpha=1 -> lasso
plot(fit.lasso,xvar='lambda',label=TRUE, col = rainbow(dim(x)[2]))
legend('topright', dimnames(x)[[2]], col =  rainbow(dim(x)[2]), lty=1, cex=1)

# Let's set lambda via CV
cv.lasso <- cv.glmnet(x,y,alpha=1,nfolds=3,lambda=lambda.grid)

bestlam.lasso <- cv.lasso$lambda.min
bestlam.lasso

plot(cv.lasso)
abline(v=log(bestlam.lasso), lty=1)

# Get the coefficients for the optimal lambda
coef.lasso <- predict(fit.lasso, s=bestlam.lasso, type = 'coefficients')[1:(p+1),]
coef.lasso

plot(fit.lasso,xvar='lambda',label=TRUE, col = rainbow(dim(x)[2]))
abline(v=log(bestlam.lasso))

# ------------------------------------------------------------------------------
# Lasso Regression         - end
# ------------------------------------------------------------------------------
```

Lasso Ridge VS. LS

```
# ------------------------------------------------------------------------------
# Compare coefficients estimates for LS, Ridge and Lasso
# ------------------------------------------------------------------------------
plot(rep(0, dim(x)[2]), coef(fm)[-1], col=rainbow(dim(x)[2]), pch=20,
     xlim=c(-1,3), ylim=c(-1,2), xlab='', ylab=expression(beta),
     axes=F)
points(rep(1, dim(x)[2]), coef.ridge[-1], col=rainbow(dim(x)[2]), pch=20)
points(rep(2, dim(x)[2]), coef.lasso[-1], col=rainbow(dim(x)[2]), pch=20)
abline(h=0, col='grey41', lty=1)
box()
axis(2)
axis(1, at=c(0,1,2), labels = c('LS', 'Ridge', 'Lasso'))
legend('topright', dimnames(x)[[2]], col =  rainbow(dim(x)[2]), pch=20, cex=1)
```

```
###-----------------------------------------------------------------------------
###-----------------------------------------------------------------------------
### Linear models
###-----------------------------------------------------------------------------
###-----------------------------------------------------------------------------
library(MASS)
library(car)
library(rgl)
library(leaps)
library(ISLR)

# ------------------------------------------------------------------------------
# ------------------------------------------------------------------------------
# PART 3 - Variables selection
# ------------------------------------------------------------------------------
# ------------------------------------------------------------------------------

data = Hitters
data = na.omit(data)
head(data)

# Rinominiamo
n = dim(data)[1]
p = dim(data)[2]-1
v = 'X1'
if(p>1){
  for(i in 2:p){
    v = c(v, paste('X',i,sep=''))
  }
}

v = c(v, 'Y')
v[19] = 'Y'
v[20] = 'X19'
v
colnames(data) = v
head(data)
attach(data)

# ------------------------------------------------------------------------------
# Best Subset Selection (exhaustive search)
# ------------------------------------------------------------------------------
# Best Subset Selection
regfit.full <- regsubsets(Y~., data=data)
summary(regfit.full)

# Best Subset Selection: we say when we stop
regfit.full <- regsubsets(Y~., data=data, nvmax=19)
summary(regfit.full)

reg.summary = summary(regfit.full)

# quale abbiamo preso
reg.summary$which

# r-squared
reg.summary$rsq

# r.adj^2
reg.summary$adjr2

# SSres (residual sum of squares)
reg.summary$rss

# PLOTS
par(mfrow=c(1,3))
plot(reg.summary$rsq,   xlab="Number of Variables", ylab="R-squared",     type="b")
plot(reg.summary$adjr2, xlab="Number of Variables", ylab="Adjusted RSq", type="b")
plot(reg.summary$rss,   xlab="Number of Variables", ylab="RSS",          type="b")
dev.off()

# We want the model with max r.adj^2:
# we extract the coefficients of that model
# Note: ind = how many coefficients has the model
ind = which.max(reg.summary$adjr2)
coef(regfit.full, ind)

# graphical table of best results
plot(regfit.full, scale="r2",    main="Exhaustive search")
plot(regfit.full, scale="adjr2", main="Exhaustive search")

# ------------------------------------------------------------------------------
# Forward and Backward Stepwise Selection
# ------------------------------------------------------------------------------
# FORWARD
regfit.fwd <- regsubsets(Y~.,data=data,nvmax=19,method="forward")
```

Subset selection (exhaustive)

Forward / Backward

F/B subset

choosing model with K-fold cross validation

```
summary(regfit.fwd)

# PLOT
par(mfrow=c(1,3))
plot(summary(regfit.fwd)$rsq,   xlab="Number of Variables", ylab="R-squared",    type="b")
plot(summary(regfit.fwd)$adjr2, xlab="Number of Variables", ylab="Adjusted RSq", type="b")
plot(summary(regfit.fwd)$rss,   xlab="Number of Variables", ylab="RSS",          type="b")
dev.off()

plot(regfit.fwd,scale="r2",main="Forward Stepwise Selection")
plot(regfit.fwd,scale="adjr2",main="Forward Stepwise Selection")

# BACKWARD
regfit.bwd <- regsubsets(Y~.,data=data,nvmax=19,method="backward")
summary(regfit.bwd)

# PLOT
par(mfrow=c(1,3))
plot(summary(regfit.bwd)$rsq,   xlab="Number of Variables", ylab="R-squared",    type="b")
plot(summary(regfit.bwd)$adjr2, xlab="Number of Variables", ylab="Adjusted RSq", type="b")
plot(summary(regfit.bwd)$rss,   xlab="Number of Variables", ylab="RSS",          type="b")
dev.off()

plot(regfit.bwd,scale="r2",main="Backward Stepwise Selection")
plot(regfit.bwd,scale="adjr2",main="Backward Stepwise Selection")

# ------------------------------------------------------------------------------
# Comparison
# ------------------------------------------------------------------------------
coef(regfit.full,7) # Exhaustive search
coef(regfit.fwd,7)  # Forward Stepwise Selection
coef(regfit.bwd,7)  # Backward Stepwise Selection

### ------------------------------------------------------------------------------
### Choosing among models using the k-fold cross-validation approach
### (exhaustive search)
### ------------------------------------------------------------------------------
k <- 10
folds <- sample(1:k,nrow(data),replace=TRUE)
table(folds)

# function that performs the prediction for regsubsets
predict.regsubsets <- function(object,newdata,id){
  form <- as.formula(object$call[[2]])
  mat <- model.matrix(form,newdata)
  coefi <- coef(object,id=id)
  xvars <- names(coefi)
  mat[,xvars]%*%coefi
}

cv.errors <- matrix(NA,k,19, dimnames=list(NULL, paste(1:19)))
for(j in 1:k){
  best.fit <- regsubsets(Y~.,data=data[folds!=j,],nvmax=19)
  for(i in 1:19){
    pred <- predict(best.fit,data[folds==j,],id=i)
    cv.errors[j,i] <- mean( (data$Y[folds==j]-pred)^2 )
  }
}
cv.errors

root.mean.cv.errors <- sqrt(apply(cv.errors,2,mean)) # average over the columns
root.mean.cv.errors

# PLOT
plot(root.mean.cv.errors,type='b')
points(which.min(root.mean.cv.errors),
       root.mean.cv.errors[which.min(root.mean.cv.errors)], col='red',pch=19)

which.min(root.mean.cv.errors)

# estimation on the full dataset
reg.best <- regsubsets(Y~.,data=data, nvmax=19)
coef(reg.best,10)
```