Sparse Regresssion

Di Prospero, Iezzi, Valentini

1/6/2021

PROBLEMA 1

Questa analisi è stata sviluppata per trovare una previsione sulla presenza o meno di cancro alla prostata. La risposta è mostrata dalla quantità di antigene specifico della prostata (lpsa); un valore elevato potrebbe significare la presenza di cancro.
Il modello sarà addestrato con una porzione dei dati denominata TRAIN e poi testato sulla parte rimanente, denominata TEST. Il metodo è definito approccio supervisionato. Questo problema si adatta con lo studio di LASSO e RIDGE perchè introducendo una costante chiamata parametro 'tuning', rappresentato con λ, c'è una diminuzione di varianza ed una soluzione alla multicollinearità delle variabili (con OLS il modello sarebbe instabile).
Questi criteri lavorano bene con grandi quantità di predittori perchè le variabili meno importanti sono schiacciate; esattamente come dice il principio del rasoio di Occam (trade-off tra complessità e accuratezza).

Con la regressione RIDGE tutte le variabili sono utilizzate e le meno importanti sono messe vicino a zero (MAI zero). Questa tecnica permette di usare tutti i parametri ma il modello finale è complesso da interpretare.

Con $\lambda = 0$, le β stimate sono le medesime di una regressione linerare semplice;
Con $\lambda = +\infty$, le β sono 0 (origine degli assi), il modello ha solo l'intercetta ($= \lambda$);
Aumentando il valore di λ, il modello è meno flessibile perchè ha meno varianza ma più distorsione;
Diminunendo il valore di λ, il modello è più flessibile perchè ha più varianza e minor distorsione.
Per scegliere il miglior lambda è possibile usare la cross-validation o AIC o BIC.

Con la regressione LASSO (least absolute shrinkage and selection operator) il problema della RIDGE è rimosso perchè quando le variabili sono inutili, i loro coefficienti β sono eliminati dal modello ponendoli uguali a zero.
La contrazione è spiegata dalla penalizzazione, essa forza i coefficienti a zero e crea un modello più interpretabile. Lo schiacciamento dei coefficienti è spiegato dalla formula che pone in valore assoluto i β, sul piano cartesiano è rappresentato da un rombo (o diamante) con baricentro sull'orgine degli assi che mostra il range di variabilità dei coefficienti. La formula romboidale favorisce il punto di tangenza delle isolinee dei termini sulla punta del rombo, comportando l'abbattimento a zero di un parametro.

Si procede caricando il dataset 'prostate cancer' e relative librerie utilizzate.

```
library(readr)
library(ISLR)
library(glmnet)
library(dplyr)
library(tidyr)
library(knitr)
```

Descrizione variabili del modello

variabili	descrizione
lpsa	level of prostate-specific antigen
lcavol	log(cancer volume)
lweight	log(prostate weight)
age	age
lbph	log(benign prostatic hyperplasia amount)
svi	seminal vesicle invasion
lcp	log(capsular penetration)
gleason	Gleason score
pgg45	percentage Gleason scores 4 or 5

Alcune considerazioni:
lcavol, lweight sono misure della prostata che avranno un rapporto lineare e positivo con lpsa.
L'età tende ad aumentare di pari passo con la probabilità di sviluppare un tumore.
La variabile svi è di tipo logica perchè indica la presenza o meno di calcoli renali.

Così è come si mostra il dataset:

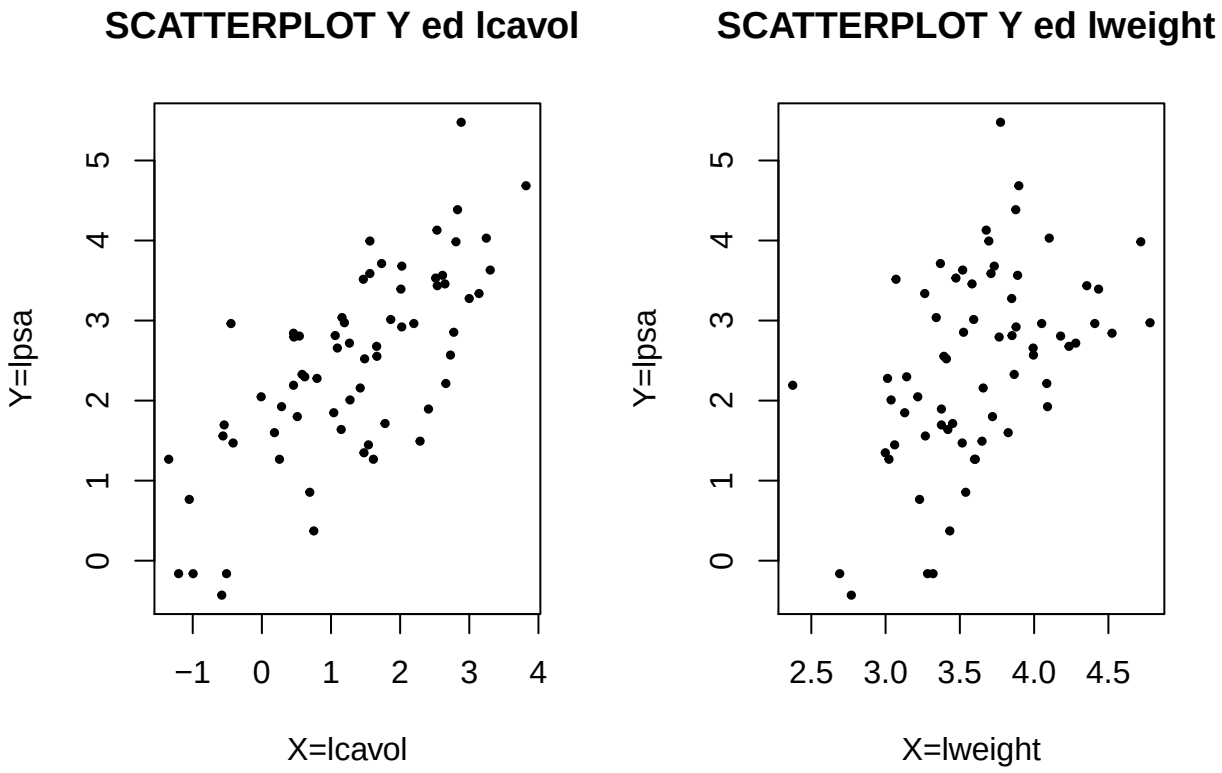
lcavol	lweight	age	lbph	svi	lcp	gleason	pgg45	lpsa	train
-0.5798185	2.769459	50	-1.386294	FALSE	-1.386294	6	0	-0.4307829	TRUE
-0.9942523	3.319626	58	-1.386294	FALSE	-1.386294	6	0	-0.1625189	TRUE
-0.5108256	2.691243	74	-1.386294	FALSE	-1.386294	7	20	-0.1625189	TRUE
-1.2039728	3.282789	58	-1.386294	FALSE	-1.386294	6	0	-0.1625189	TRUE
0.7514161	3.432373	62	-1.386294	FALSE	-1.386294	6	0	0.3715636	TRUE
-1.0498221	3.228826	50	-1.386294	FALSE	-1.386294	6	0	0.7654678	TRUE

La variabile SVI è stata posta come logica.

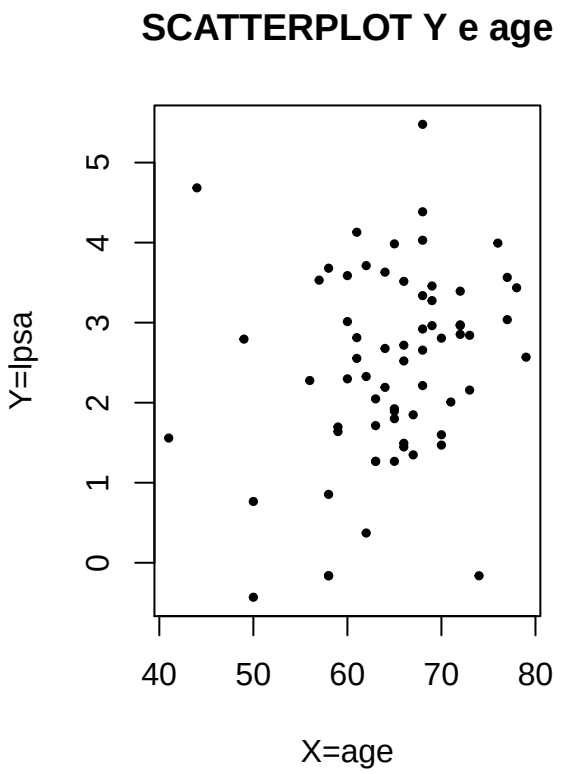
Suddivisione matrice dati in train e test set

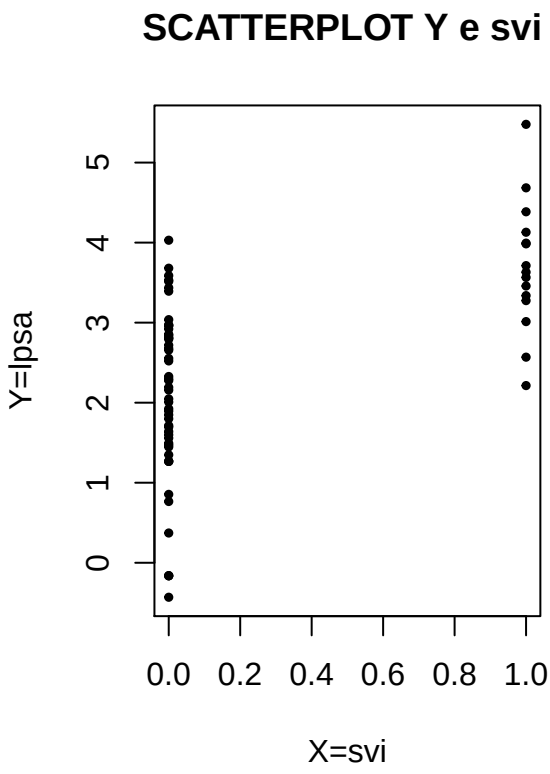
```
#train
train <- subset(prostate, train == "TRUE")
train_FREE <- train[,-10]
x_train_matrix <- train_FREE %>%
  select(-lpsa) %>%
  as.matrix()
y_train <-  train_FREE %>%
  select(lpsa) %>%
  scale(center = TRUE, scale = FALSE) %>%
  as.matrix()
#test
test <- subset(prostate, train == "FALSE")
test_FREE <- test[,-10]
x_test_matrix <- test_FREE %>%
  select(-lpsa) %>%
  as.matrix()
y_test <- test_FREE %>%
  select(lpsa) %>%
  scale(center = TRUE, scale = FALSE)
```

Analisi esplorativa



Come già accennato sopra, le due variabili 'lcavol' ed 'lweight' hanno un rapporto lineare e positivo con lpsa.





Osservando il rapporto con 'age' si smentisce quanto dedotto sopra (nella descrizione delle variabili) perchè c'è un raggruppamento di valori non ben definibile da una relazione lineare.
La var. 'svi' è mostrata per chiarezza rispetto alla decisione assunta di porla come logica.

a)

Regressione linerare su train.

```
lm_TRAIN<- lm(lpsa~., data = train_FREE)
```

I coefficienti stimati:

```
##
## Call:
## lm(formula = lpsa ~ ., data = train_FREE)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.64870 -0.34147 -0.05424  0.44941  1.48675
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.429170   1.553588   0.276  0.78334
## lcavol       0.576543   0.107438   5.366 1.47e-06 ***
```

```
## lweight      0.614020   0.223216   2.751  0.00792 **
## age         -0.019001   0.013612  -1.396  0.16806
## lbph         0.144848   0.070457   2.056  0.04431 *
## sviTRUE      0.737209   0.298555   2.469  0.01651 *
## lcp         -0.206324   0.110516  -1.867  0.06697 .
## gleason     -0.029503   0.201136  -0.147  0.88389
## pgg45        0.009465   0.005447   1.738  0.08755 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7123 on 58 degrees of freedom
## Multiple R-squared:  0.6944, Adjusted R-squared:  0.6522
## F-statistic: 16.47 on 8 and 58 DF,  p-value: 2.042e-12
```

L'OLS generale mostra un R-squared di 0.7.
I coefficienti rilevanti (α>0.05) sono: 'lcavol', 'lweight', 'lbph', 'sviTRUE' (variabile logica).

MSE su TEST	MSE su TRAIN
0.521274	0.4391998

Nella prima colonna il MSE del modello applicato sul test, nell'altra sul train.
Si nota un incremento di errore previsionale con il test, ciò può essere causato da un overfitting perchè il linear model utilizza ed interpreta tutte le variabili inserite.
Inoltre la multicollinaerità tra le variabili crea instabilità del modello OLS perchè comporta una modesta significatività delle variabili ed un R quadro elevato.
A dimostrazione di questo è opportuno applicare il fattore di inflazione della varianza su lm_TRAIN

```
library(car)
kable(vif(lm_TRAIN), col.names = "Mean squared error")
```

	Mean squared error
lcavol	2.318496
lweight	1.472295
age	1.356604
lbph	1.383429
svi	2.045313
lcp	3.117451
gleason	2.644480
pgg45	3.313289

Questa funzione applica la seguente formula:

$$VIF = \frac{1}{1 - R_i^2}$$

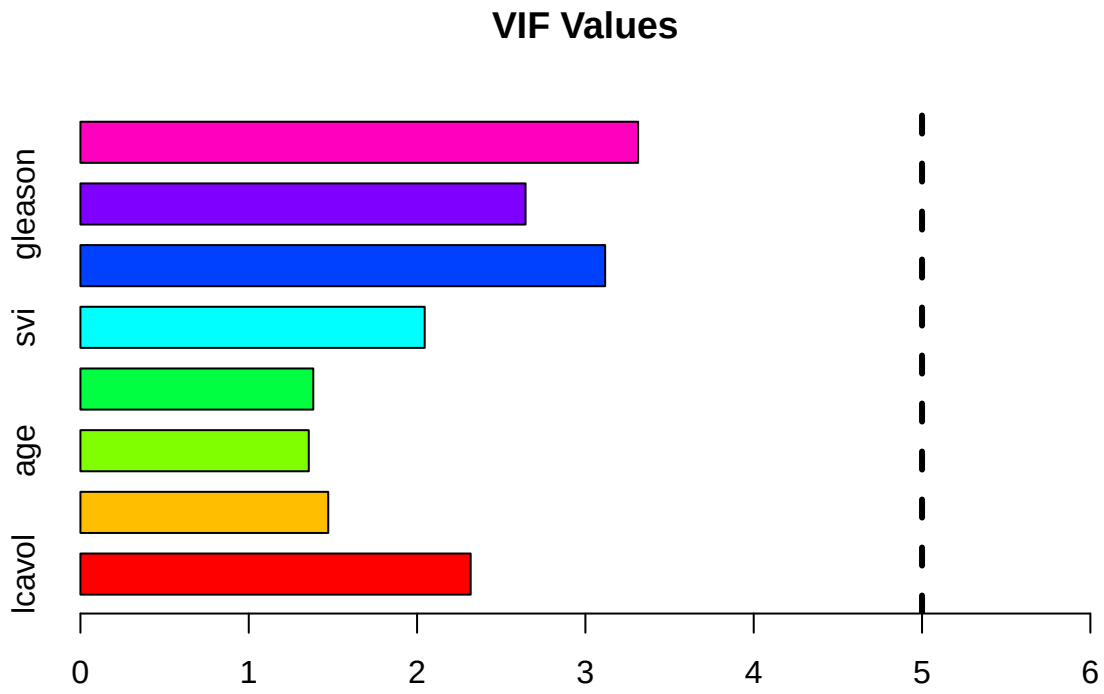
Il VIF è calcolato per ciascuna variabile del modello, ponendo la variabile in esame come fosse la 'y' e tutte le altre variabili come predittori. La formula per il calcolo ols del primo coefficiente è la seguente:

$$X_1 = \alpha_0 + \alpha 2 X_2 + \alpha_3 X_3 + ... + \alpha_k X_k + e$$

Successivamente si calcolano le beta stimate e relativo VIF. Nella formula del VIF l'R-squared è calcolato per la formula soprastante. Se ne conclude che con un R^2 elevato, ad esempio 0.8, darà come risultato un VIF pari a 5. In conclusione un R^2 elevato è indice di multicollinearità.

- VIF vicino 1 indica che non c'è correlazione
- VIF compreso tra 1 e 5 indica moderata correlazione, ma usualmente non è severa a tal punto da richiedere attenzione
- VIF maggiore di 5 indica che c'è elevata multicollinearità, quindi l'OLS è inaffidabile

Analizzando l'output lpc e pgg45 sono le variabili che hanno una mediocre correlazione con le altre variabili, quindi distorcono l'OLS e andrebbero eliminate.



b)

BIC selection

Calcolato l'OLS generale, ora è necessario trovare il miglior OLS usando il criterio BIC. Si inizia con una sola variabile significante, lcavol, e si aggiungono una alla volta tutte le altre secondo il criterio 'forward'.

```
lm_lcavol <- lm(lpsa~lcavol, data =train_FREE)
summary(lm_lcavol)
```

```
##
## Call:
```

```
## lm(formula = lpsa ~ lcavol, data = train_FREE)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.6802 -0.4240  0.1684  0.5392  1.9070
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.51630    0.14772  10.264 3.12e-15 ***
## lcavol       0.71264    0.08199   8.692 1.73e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8277 on 65 degrees of freedom
## Multiple R-squared:  0.5375, Adjusted R-squared:  0.5304
## F-statistic: 75.55 on 1 and 65 DF,  p-value: 1.733e-12
```

L R-squared è 0.54. Consultando lm_TRAIN, si aggiunge lweight

```
lm_lcavol_lweight<- lm(lpsa~lcavol+
                         lweight,
                       data = train_FREE)
summary(lm_lcavol_lweight)
```

```
##
## Call:
## lm(formula = lpsa ~ lcavol + lweight, data = train_FREE)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.58852 -0.44174  0.01304  0.52613  1.93127
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.04944    0.72904  -1.439 0.154885
## lcavol       0.62761    0.07906   7.938 4.14e-11 ***
## lweight      0.73838    0.20613   3.582 0.000658 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7613 on 64 degrees of freedom
## Multiple R-squared:  0.6148, Adjusted R-squared:  0.6027
## F-statistic: 51.06 on 2 and 64 DF,  p-value: 5.54e-14
```

R-squared aumenta ed entrambe le variabili sono significative. Si aggiunge svi.

```
lm_lcavol_lweight_svi<- lm(lpsa~lcavol+
                             lweight+
                             svi,
                           data = train_FREE)
summary(lm_lcavol_lweight_svi)
```

```
##
```

```
## Call:
## lm(formula = lpsa ~ lcavol + lweight + svi, data = train_FREE)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.69494 -0.46058 -0.04485  0.49149  1.68289
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.02278    0.71296  -1.435 0.156362
## lcavol       0.51999    0.09442   5.507 7.16e-07 ***
## lweight      0.73680    0.20155   3.656 0.000525 ***
## sviTRUE      0.53790    0.27093   1.985 0.051458 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7444 on 63 degrees of freedom
## Multiple R-squared:  0.6374, Adjusted R-squared:  0.6202
## F-statistic: 36.92 on 3 and 63 DF,  p-value: 6.81e-14
```

Si aggiunge lbph

```
lm_lcavol_lweight_svi_lbph<- lm(lpsa~lcavol+
                                  lweight+
                                  svi+
                                  lbph,
                                data = train_FREE) #+lbph
summary(lm_lcavol_lweight_svi_lbph)
```

```
##
## Call:
## lm(formula = lpsa ~ lcavol + lweight + svi + lbph, data = train_FREE)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.8709 -0.3903 -0.0172  0.5676  1.4227
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.32592    0.77998  -0.418   0.6775
## lcavol       0.50552    0.09256   5.461 8.85e-07 ***
## lweight      0.53883    0.22071   2.441   0.0175 *
## sviTRUE      0.67185    0.27323   2.459   0.0167 *
## lbph         0.14001    0.07041   1.988   0.0512 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7275 on 62 degrees of freedom
## Multiple R-squared:  0.6592, Adjusted R-squared:  0.6372
## F-statistic: 29.98 on 4 and 62 DF,  p-value: 6.911e-14
```

Il P-value di svi decresce mentre per lweight aumenta. lbph non è significativo.
Si procede aggiungendo lcp.

```r
lm_lcavol_lweight_svi_lbph_lcp<- lm(lpsa~lcavol+
                                      lweight+
                                      svi+
                                      lbph+
                                      lcp,
                                    data = train_FREE)
summary(lm_lcavol_lweight_svi_lbph_lcp)
```

```
##
## Call:
## lm(formula = lpsa ~ lcavol + lweight + svi + lbph + lcp, data = train_FREE)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.82988 -0.40161  0.02167  0.50989  1.42593
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.40619    0.78486  -0.518   0.6067
## lcavol       0.55698    0.10690   5.210 2.36e-06 ***
## lweight      0.52904    0.22107   2.393   0.0198 *
## sviTRUE      0.79660    0.30248   2.634   0.0107 *
## lbph         0.13548    0.07061   1.919   0.0597 .
## lcp         -0.09613    0.09975  -0.964   0.3390
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7279 on 61 degrees of freedom
## Multiple R-squared:  0.6643, Adjusted R-squared:  0.6368
## F-statistic: 24.14 on 5 and 61 DF,  p-value: 2.599e-13
```

Si procede aggiungendo pgg45

```r
lm_lcavol_lweight_svi_lbph_lcp_pgg45<- lm(lpsa~lcavol+
                                            lweight+
                                            svi+
                                            lbph+
                                            lcp+
                                            pgg45,
                                          data = train_FREE)
summary(lm_lcavol_lweight_svi_lbph_lcp_pgg45)
```

```
##
## Call:
## lm(formula = lpsa ~ lcavol + lweight + svi + lbph + lcp + pgg45,
##     data = train_FREE)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.69209 -0.33444 -0.05102  0.53576  1.37947
##
## Coefficients:
```

```
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.728972   0.788328  -0.925   0.3588
## lcavol       0.549778   0.104846   5.244 2.15e-06 ***
## lweight      0.563106   0.217436   2.590   0.0120 *
## sviTRUE      0.756355   0.297239   2.545   0.0135 *
## lbph         0.125979   0.069389   1.816   0.0744 .
## lcp         -0.190825   0.110076  -1.734   0.0881 .
## pgg45        0.007541   0.004029   1.872   0.0661 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7134 on 60 degrees of freedom
## Multiple R-squared:  0.6828, Adjusted R-squared:  0.6511
## F-statistic: 21.53 on 6 and 60 DF,  p-value: 2.611e-13
```

lcp diminuisce e tutti i coefficienti sono significanti, solo gli ultimi tre hanno $\alpha > 0.05$. Si aggiunge l'ultima variabile age.

```
lm_lcavol_lweight_svi_lbph_lcp_pgg45_age<- lm(lpsa~lcavol+
                                                lweight+
                                                svi+
                                                lbph+
                                                lcp+
                                                pgg45+
                                                age,
                                              data = train_FREE)
summary(lm_lcavol_lweight_svi_lbph_lcp_pgg45_age)
```

```
##
## Call:
## lm(formula = lpsa ~ lcavol + lweight + svi + lbph + lcp + pgg45 +
##     age, data = train_FREE)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.65425 -0.34471 -0.05615  0.44380  1.48952
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.259062   1.025170   0.253   0.8014
## lcavol       0.573930   0.105069   5.462 9.88e-07 ***
## lweight      0.619209   0.218560   2.833   0.0063 **
## sviTRUE      0.741781   0.294451   2.519   0.0145 *
## lbph         0.144426   0.069812   2.069   0.0430 *
## lcp         -0.205417   0.109424  -1.877   0.0654 .
## pgg45        0.008945   0.004099   2.182   0.0331 *
## age         -0.019480   0.013105  -1.486   0.1425
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7064 on 59 degrees of freedom
## Multiple R-squared:  0.6943, Adjusted R-squared:  0.658
## F-statistic: 19.14 on 7 and 59 DF,  p-value: 4.496e-13
```

E' il momento di scegliere il miglior modello con il BIC

	df	BIC
lm_TRAIN	10	177.0570
lm_lcavol	3	175.3782
lm_lcavol_lweight	4	167.3397
lm_lcavol_lweight_svi	5	167.4783
lm_lcavol_lweight_svi_lbph	6	167.5408
lm_lcavol_lweight_svi_lbph_lcp	7	170.7331
lm_lcavol_lweight_svi_lbph_lcp_pgg45	8	171.1359
lm_lcavol_lweight_svi_lbph_lcp_pgg45_age	9	172.8772

```
## [1] 3
```

Il valore mostrato da 'min' è riferito alla riga, quindi a 'lm_lcavol_lweight'.
Il più basso BIC si ha con la regressione comprensiva di variabili esplicative $lcavol + lweight$
Seguono codici che mostrano i coefficienti ed il test error:

	Linear model
(Intercept)	-1.0494396
lcavol	0.6276074
lweight	0.7383751

MSE su Test	MSE su Train
0.4924823	0.5536096

La formula utilizzata per calcolare l'errore quadratico medio è:

$$MSE = \frac{1}{n}\Sigma(y_i - f(x_i))^2$$

L'MSE ottenuto sul test set è di 0.49, di 0.55 sul train.

MSE_best_lm_on_TEST	MSE_lm_TRAIN_on_TEST
0.4924823	0.521274

Utilizzando il test set, si mostra l'MSE ottenuto con il miglior modello selezionato dal BIC e l'MSE con il modello comprensivo di tutte le variabili.
Si nota che il modello selezionato con il BIC ottiene un minor errore sulla previsione, quindi evince come la, seppure non eccesiva, multicollineraità presente tra le variabili distorce le capacità previsive del modello globale.

R2_TRAIN	R2_best
0.6943712	0.614756

Ulteriore riprova di quanto enunciato sopra è il confronto di R-squared. Sebbene quello della regressione globale (colonna 1) sia più elevato, nella previsione funziona meglio quello più semplice, con due variabili

indipendenti, anche se ha un R2 più basso.
Questa evidenza empirica mostra la necessità di procedere con regressioni RIDGE e LASSO.

c)

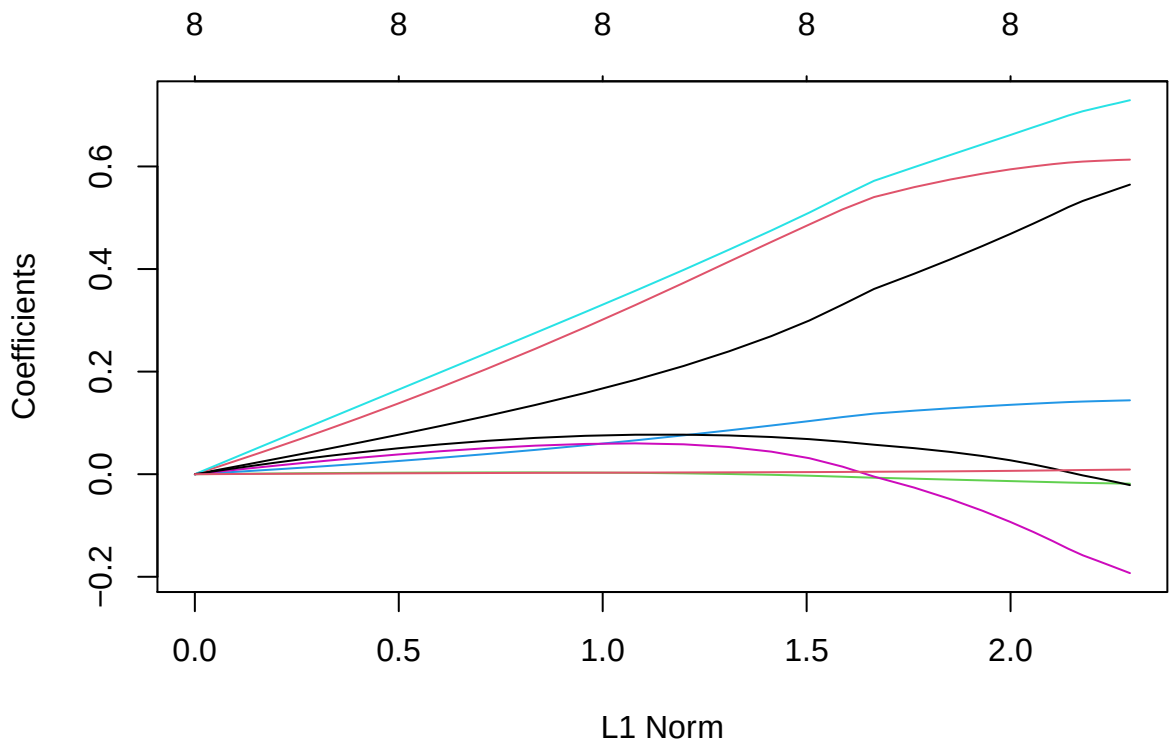
Ridge

Si inizia applicando la ridge al training set.

```
set.seed(1)
grid = 10^seq(10, -2, length = 100)
ridge_mod <- glmnet(x_train_matrix , y_train, alpha = 0, lambda = grid)
dim(coef(ridge_mod))
```

```
## [1]   9 100
```

Prima di tutto si crea una griglia di 100 valori per λ.
L'output di `dim` mostra 10 coefficienti perchè ci sono 9 regressori più l'intercetta, e 100 colonne per differenti valori di λ.

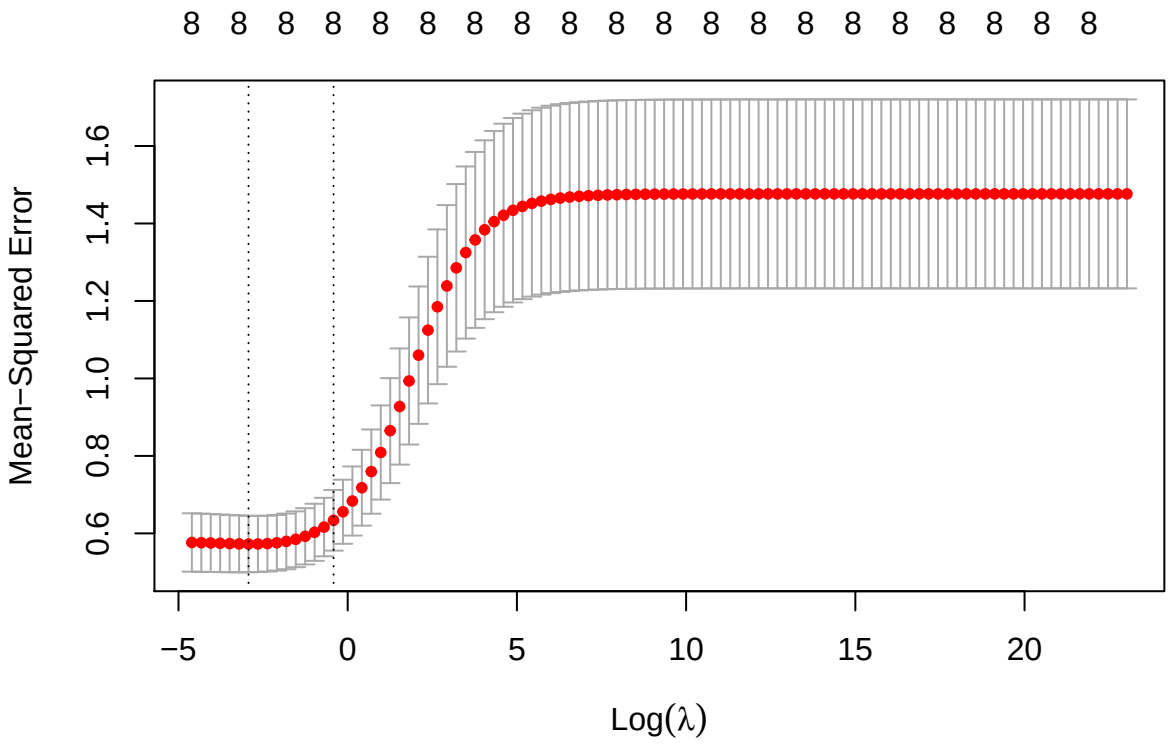


Dal grafico si notano due coefficienti che restano vicini allo zero.

```
cv.out = cv.glmnet(x_train_matrix, y_train, alpha = 0, lambda = grid)
best_lambda = cv.out$lambda.min
best_lambda
```

```
## [1] 0.05336699
```

```
plot(cv.out)
```



Si mostra il miglior lambda selezionato con la cross-validazione per RIDGE: 0.05336699 (-2,93056)

Ora si procede con il determinare il miglior λ utilizzando la tecnica BIC:

```
n_Ridge <- ridge_mod$nobs
MSE_Ridge <- deviance(ridge_mod)/n_Ridge
df_Ridge <- ridge_mod$df
BIC <- log(n_Ridge)*df_Ridge + n_Ridge*log(MSE_Ridge)
BIC
```

```
##   [1]  57.930602  57.930602  57.930602  57.930602  57.930602  57.930602
##   [7]  57.930602  57.930602  57.930602  57.930602  57.930602  57.930601
##  [13]  57.930601  57.930601  57.930601  57.930600  57.930600  57.930599
##  [19]  57.930598  57.930596  57.930595  57.930592  57.930589  57.930585
##  [25]  57.930579  57.930572  57.930562  57.930549  57.930532  57.930509
##  [31]  57.930480  57.930440  57.930388  57.930319  57.930228  57.930108
##  [37]  57.929948  57.929738  57.929460  57.929092  57.928606  57.927964
##  [43]  57.927114  57.925991  57.924507  57.922545  57.919951  57.916523
##  [49]  57.911991  57.906001  57.898084  57.887620  57.873791  57.855517
##  [55]  57.831372  57.799474  57.757342  57.701709  57.628275  57.531391
##  [61]  57.403954  57.235890  57.014816  56.724473  56.343944  55.846573
##  [67]  55.198788  54.359010  53.276870  51.893223  50.141617  47.952064
```

```
##  [73]  45.258165  42.008062  38.178833  33.791890  28.924632  23.712178
##  [79]  18.334630  12.990505   7.863917   3.093915  -1.223246  -5.052121
##  [85]  -8.388575 -11.254387 -13.669699 -15.668676 -17.273436 -18.525789
##  [91] -19.466214 -20.145713 -20.620379 -20.939874 -21.148774 -21.281841
##  [97] -21.364759 -21.415051 -21.445807 -21.464144
```

La numerosità dei parametri ed i gradi di libertà sono i medesimi per la natura stessa del Ridge che non elimina parametri insignificanti.

```
min(BIC)
```

```
## [1] -21.46414
```

```
which.min(BIC)
```

```
## [1] 100
```

```
ridge_mod$lambda[100]
```

```
## [1] 0.01
```

Il BIC più basso è l'ultimo, quindi quello in posizione 100 a cui corrisponde il relativo lambda pari a 0.01.

Ora gli errori medi quadratici dei modelli sul train e test set

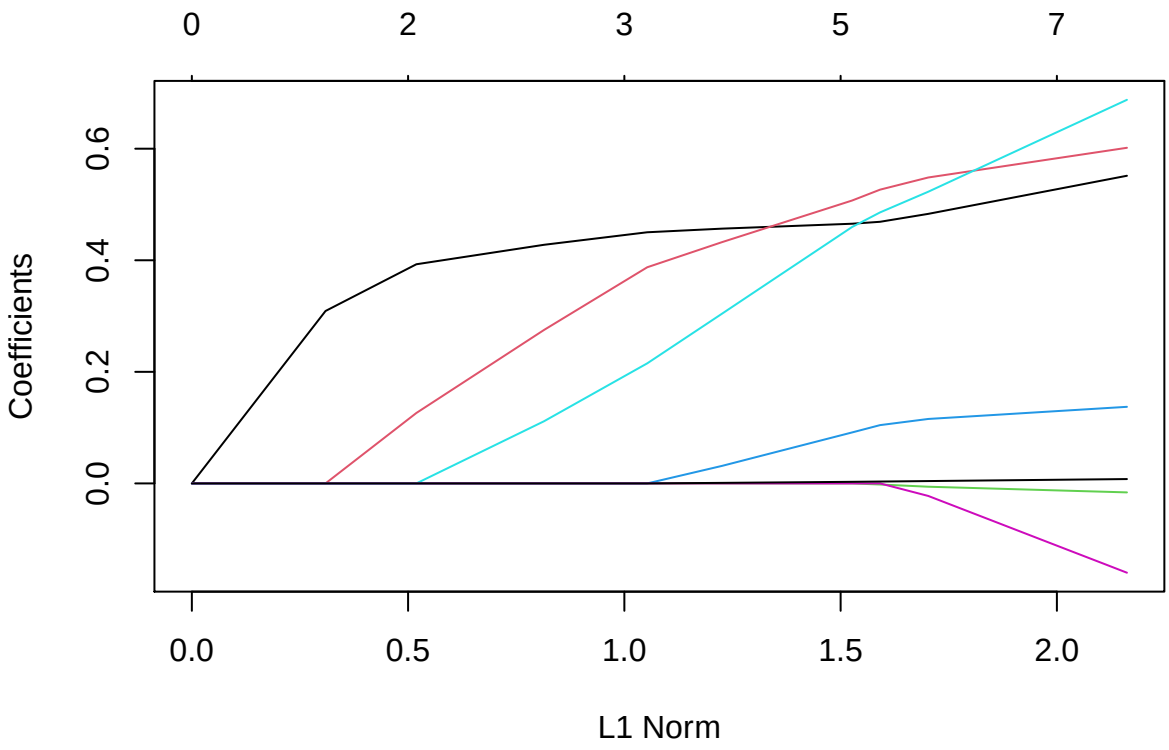
MSE_Ridge_bic	MSE_Ridge_cv
0.4393684	0.4428203

Gli errori di previsione sul train sono simili, il Ridge con lambda calcolato con BIC è sensibilmente migliore.

MSE_Ridge_bic_TEST	MSE_Ridge_cv_TEST
0.5160845	0.5016679

Anche in questo caso gli errori sono similari. Si nota un generale aumento rispetto alla predizione sul train e una inversione di tendenza con il lambda calcolato con cross-validazione che dà stime migliori, seppur di poco, sul test set rispetto al modello con metodo BIC.

Lasso



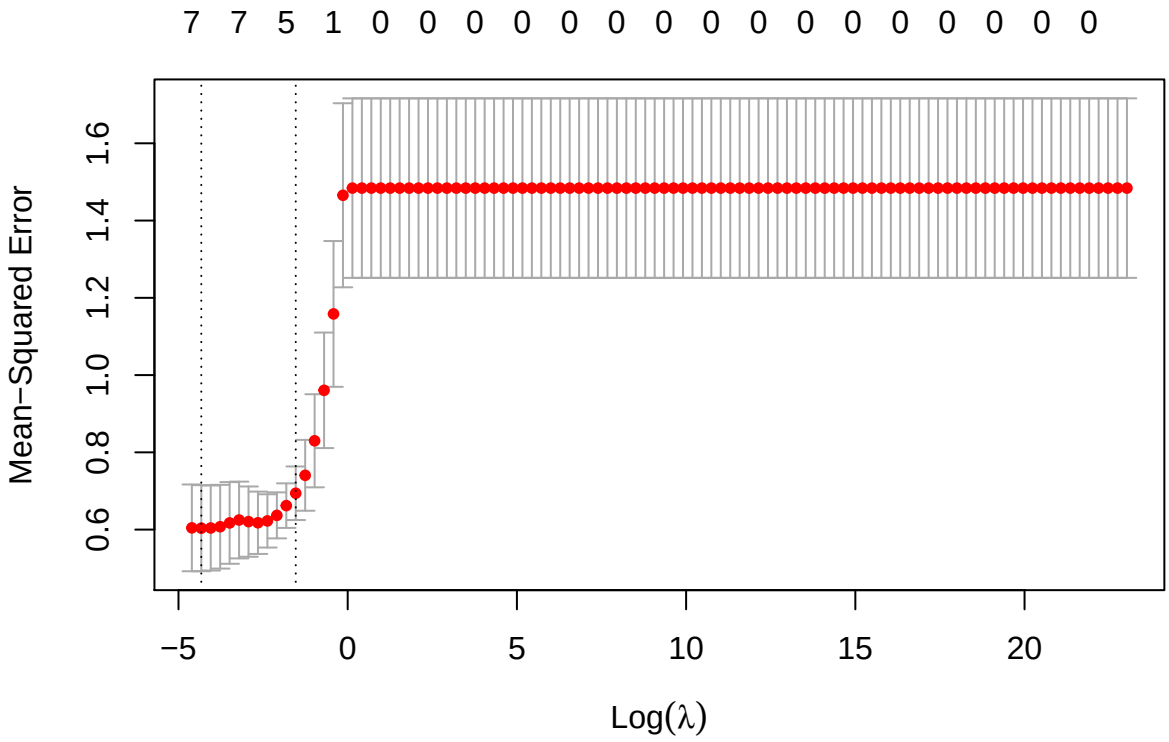
Ora si procede con il calcolo del modello LASSO e selezione dei migliori λ con la cross-validazione e BIC.

```
cv.out.lasso = cv.glmnet(x_train_matrix, y_train, alpha = 1, lambda = grid)
best_lambda_lasso = cv.out.lasso$lambda.min
best_lambda_lasso
```

```
## [1] 0.01321941
```

Il miglior λ ottenuto con la cross-validation è di circa 0.01.

```
plot(cv.out.lasso)
```



Ora si passa al calcolo del miglior lambda usando il metodo BIC.

```
n_lasso <- lasso_mod$nobs
MSE_lasso <- deviance(lasso_mod)/n_lasso
df_lasso <- lasso_mod$df
BIC_lasso <- log(n_lasso)*df_lasso + n_lasso*log(MSE_lasso)
```

```
min(BIC_lasso)
```

```
## [1] -26.45448
```

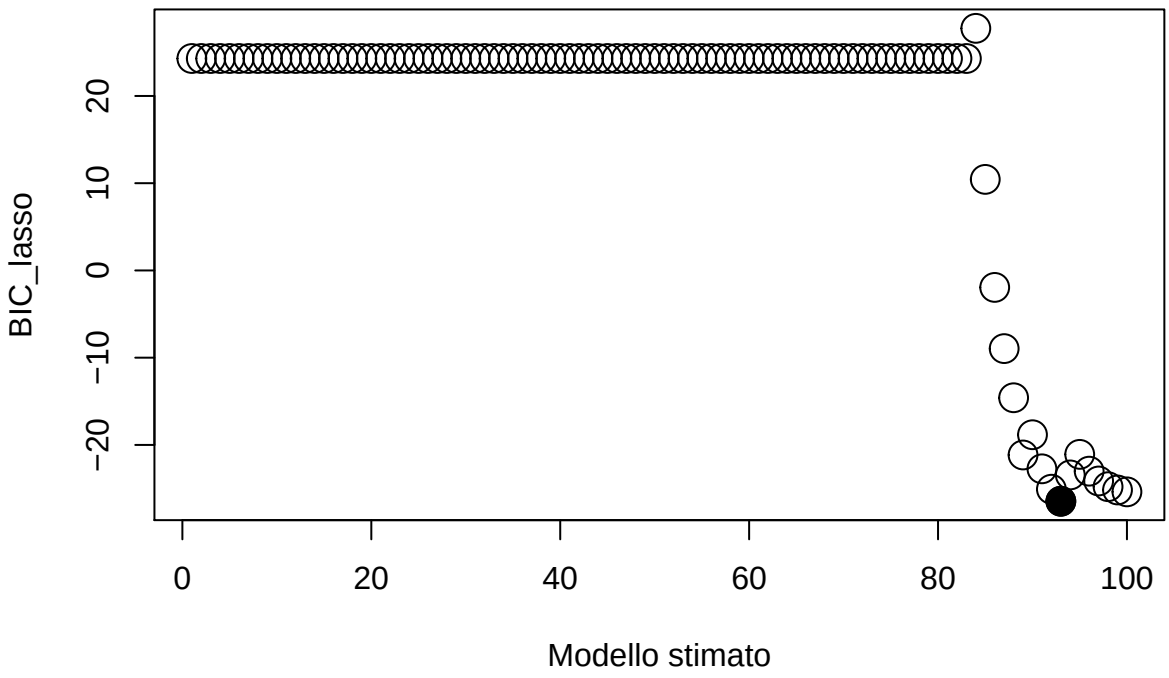
Il BIC più basso è -26.45448 e sulla griglia è posizionato:

```
## [1] 93
```

```
lambda_lasso_bic
```

```
## [1] 0.07054802
```

Il relativo lambda è pari a 0.07054802



Nel grafico sono riportati i modelli stimati ed i loro BIC.

Si procede con il confronto tra il miglior MSE tra i due modelli

MSE_Lasso_bic	MSE_Lasso_cv
0.4923267	0.4426998

Sul train il modello migliore è quello ottenuto con la cross-validation.

MSE_Lasso_bic_TEST	MSE_Lasso_cv_TEST
0.4563866	0.4944581

Qui si inverte ed il miglior modello è quello ottenuto con il BIC.

Si arriva alla fase finale. Vengono presi i modelli con MSE inferiori su TEST SET e se ne mostrano i coefficienti.

Ed ecco il confronto tra tutti i migliori modelli.

	Mean Squared Error
MSE_lm_TRAIN_on_TEST	0.5212740
MSE_best_lm_on_TEST	0.4924823
MSE_Ridge_cv_TEST	0.5016679
MSE_Lasso_bic_TEST	0.4563866

Il modello che meglio apprende dal training, quindi che prevede meglio, è LASSO con lambda pari a circa 0,07; calcolato utilizzando il BIC.
Seguono regressione lineare con due parametri, e Ridge calcolata con cross-validation.

```
kable(matriceRis)
```

	beta lm TRAIN	beta lm BIC	beta ridge CV	beta lasso BIC
Intercept	0.4291701	-1.0494396	-2.2587073	-2.6343785
lcavol	0.5765432	0.6276074	0.5207802	0.4654317
lweight	0.6140200	0.7383751	0.6072888	0.5069927
age	-0.0190010	0.0000000	-0.0163076	0.0000000
lbph	0.1448481	0.0000000	0.1406678	0.0914129
svi	0.7372086	0.0000000	0.6996994	0.4597639
lcp	-0.2063242	0.0000000	-0.1454748	0.0000000
gleason	-0.0295029	0.0000000	0.0038402	0.0000000
pgg45	0.0094652	0.0000000	0.0077933	0.0028614

Ricordando che il miglior modello è quello ottenuto con lasso (ultima colonna), si analizzano i coefficienti.
'gleason', 'lcp' e 'age' sono inutili ai fini previsionali e sono schiacciati pari a zero.
'lcavol' ed 'lweight' sono importanti nella previsione e vengono mantenuti da tutti i modelli, ovviamente con valori positivi.

PROBLEMA 2

Introduzione al problema

Le simulazioni costituiscono uno strumento molto valido per comprendere le caratteristiche di una determinata metodologia statistica,e per valutarne il suo comportamento (ossia, i punti di forza e di debolezza della metodologia adottata).

Generiamo la matrice quadrata H di dimensione 10000 x 10000, seguendo la logica mostrata in figura.

$$\boldsymbol{H} = \begin{bmatrix} 1/\sqrt{n} & 1/\sqrt{2} & 1/\sqrt{6} & 1/\sqrt{12} & \dots & 1/\sqrt{n(n-1)} \\ 1/\sqrt{n} & -1/\sqrt{2} & 1/\sqrt{6} & 1/\sqrt{12} & \dots & 1/\sqrt{n(n-1)} \\ 1/\sqrt{n} & 0 & -2/\sqrt{6} & 1/\sqrt{12} & \dots & 1/\sqrt{n(n-1)} \\ 1/\sqrt{n} & 0 & 0 & -3/\sqrt{12} & \dots & 1/\sqrt{n(n-1)} \\ \vdots & 0 & 0 & 0 & \dots & \vdots \\ 1/\sqrt{n} & 0 & 0 & \dots & \dots & -(n-1)/\sqrt{n(n-1)} \end{bmatrix}$$

```
H = matrix(nrow = n, ncol = n)
for(i in 1:n) {
       H[i,1]=1/sqrt(n)
       }
for (i in 1:n-1) {
       for (j in (i+1):n) {
```

```
        H[i,j]= 1/sqrt(j*(j-1))
       }
  }
for (i in 2:n) {
  H[i,i]= -(i-1)/sqrt(i*(i-1))
  }
for (i in 1:n) {
  for (j in 1:n) {
    if(is.na( H[i,j]))
    {
    H[i,j]= 0
    }
  }
  }
dim(H)
```

```
## [1] 10000 10000
```

Ora generiamo la matrice X di dimensione 10000 x 1000, prendendo le prime 1000 colonne della matrice H. Generiamo anche il vettore dei coefficienti reali ß e il termine di errore generato da una distibuzione normale con media = 0 e deviazione standard = 1

```
X= matrix(rnorm(H[,1:p]), nrow = n, ncol = p)
dim(X)
```

```
## [1] 10000  1000
```

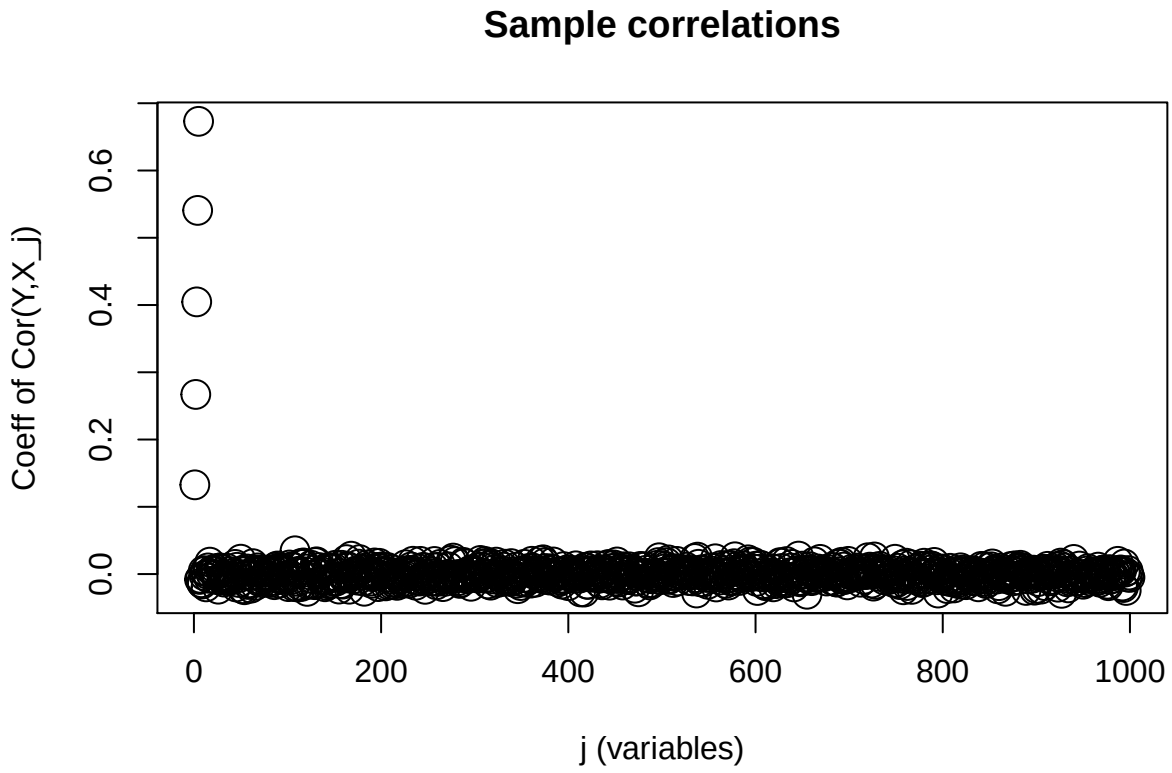
```
beta      <- rep(0,p)
beta[1:5] <- 1:5
e= rnorm(n)
```

Dopo aver stanndardizzato la matrice X, simuliamo il modello di regressione.

```
X <- scale(X)
Xb <- X%*%beta
Y <- X%*%beta+e
Y <- Y-mean(Y)
```

Il grafico delle correlazioni seguente mostra come ciascun predittore si va a correlare con la nostra Y. Solo per i primi 5 si evidenziano legami significativi.

```
plot(cor(X,Y),xlab="j (variables)",ylab="Coeff of Cor(Y,X_j)",main="Sample correlations",cex=2)
```



Usiamo il metodo OLS per stimare il modello di regressione e mostriamo i coefficienti significativi allo 0.05.

```
ols <- lm(Y~X)
beta_ols <- ols$coef[-1]
round(summary(ols)$coef[summary(ols)$coef[,4] < .06, 1:4], digits = 7)
```

```
##        Estimate Std. Error    t value  Pr(>|t|)
## X1    0.9818859  0.0104564  93.902675 0.0000000
## X2    2.0061491  0.0104185 192.556343 0.0000000
## X3    3.0127981  0.0104128 289.335104 0.0000000
## X4    4.0048383  0.0104191 384.374591 0.0000000
## X5    5.0142874  0.0104399 480.298894 0.0000000
## X60  -0.0203254  0.0104124  -1.952033 0.0509652
## X63   0.0214248  0.0104049   2.059102 0.0395132
## X71   0.0269904  0.0104349   2.586567 0.0097093
## X84   0.0213098  0.0104274   2.043632 0.0410190
## X115 -0.0204516  0.0104171  -1.963272 0.0496453
## X122 -0.0202837  0.0104136  -1.947805 0.0514693
## X132 -0.0222907  0.0104257  -2.138052 0.0325394
## X133 -0.0204523  0.0104241  -1.962017 0.0497913
## X191  0.0196856  0.0104229   1.888685 0.0589661
## X200  0.0219425  0.0104379   2.102190 0.0355643
## X203  0.0291643  0.0104450   2.792169 0.0052467
## X204  0.0260152  0.0104152   2.497810 0.0125140
## X210 -0.0220024  0.0104147  -2.112639 0.0346591
## X214  0.0220182  0.0104997   2.097025 0.0360192
```

```
## X228 -0.0213923  0.0103902  -2.058891 0.0395334
## X277  0.0197554  0.0104555   1.889462 0.0588620
## X278  0.0222603  0.0104123   2.137896 0.0325520
## X306  0.0200560  0.0103860   1.931066 0.0535063
## X308  0.0216806  0.0104157   2.081534 0.0374132
## X314 -0.0209504  0.0104306  -2.008561 0.0446136
## X326 -0.0279864  0.0104170  -2.686615 0.0072313
## X328  0.0229097  0.0104044   2.201923 0.0276960
## X333 -0.0217508  0.0104297  -2.085474 0.0370544
## X334  0.0248130  0.0104130   2.382880 0.0171984
## X359 -0.0260214  0.0104354  -2.493564 0.0126646
## X372 -0.0225538  0.0104105  -2.166457 0.0303024
## X383  0.0235066  0.0104289   2.253989 0.0242208
## X402 -0.0214227  0.0104276  -2.054427 0.0399632
## X404  0.0275486  0.0104352   2.639960 0.0083059
## X409 -0.0289325  0.0104275  -2.774633 0.0055378
## X432  0.0250035  0.0103994   2.404312 0.0162230
## X433 -0.0266618  0.0103709  -2.570836 0.0101613
## X444 -0.0235691  0.0104453  -2.256423 0.0240681
## X464 -0.0210685  0.0104216  -2.021627 0.0432444
## X514  0.0263621  0.0103886   2.537593 0.0111785
## X554  0.0321459  0.0104577   3.073913 0.0021190
## X569 -0.0293882  0.0104039  -2.824728 0.0047425
## X594  0.0218084  0.0104519   2.086543 0.0369575
## X595  0.0210331  0.0103882   2.024710 0.0429266
## X616  0.0226938  0.0104198   2.177949 0.0294357
## X651 -0.0204992  0.0103977  -1.971520 0.0486950
## X654  0.0212004  0.0104393   2.030822 0.0423024
## X657  0.0206987  0.0104040   1.989496 0.0466767
## X670 -0.0226925  0.0104168  -2.178455 0.0293981
## X673 -0.0233962  0.0104227  -2.244743 0.0248088
## X685  0.0222916  0.0104408   2.135046 0.0327842
## X719  0.0263304  0.0104491   2.519875 0.0117568
## X720  0.0212789  0.0104344   2.039309 0.0414483
## X782  0.0234930  0.0104284   2.252788 0.0242965
## X815 -0.0248126  0.0103853  -2.389196 0.0169058
## X825 -0.0259548  0.0103681  -2.503343 0.0123202
## X859 -0.0255542  0.0104055  -2.455845 0.0140742
## X861  0.0197997  0.0104160   1.900904 0.0573465
## X947 -0.0225337  0.0103810  -2.170662 0.0299828
## X950 -0.0271776  0.0104150  -2.609463 0.0090834
```

La soluzione ai minimi quadrati non restituisce una buona soluzione in quanto riconosce come molto significativi i legami con le prime 5 variabili, ma riconosce come significativi anche i legami con altre variabili che noi sappiamo invece essere pari a 0 nel nostro modello.

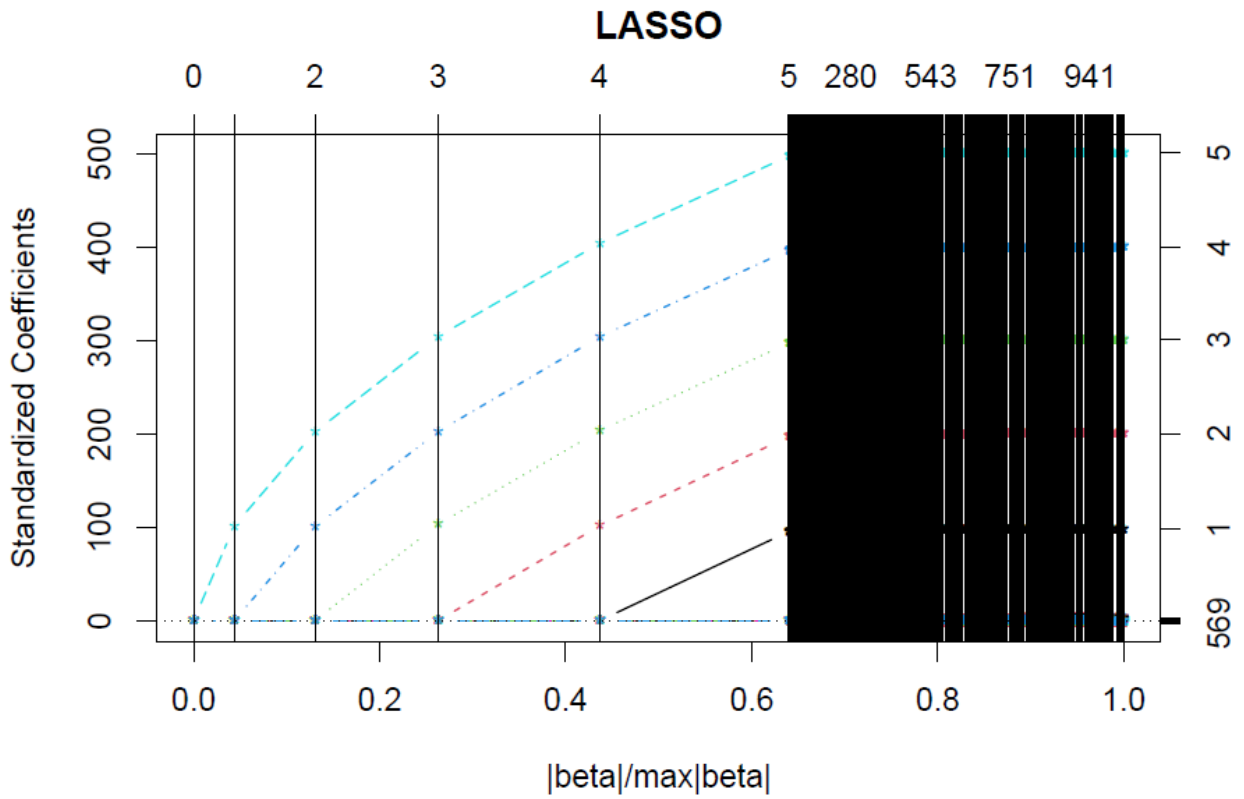
Lasso

Lars

Carichiamo la libreria lars e calcoliamo la soluzione lasso.

```
library(knitr)
library(lars)
lasso  <- lars(X,Y)
```

```
plot(lasso)
```



Nel grafico precedente, tanto più mi sposto verso la fine tanto più vado verso una soluzione OLS, in quanto il termine di regolarizzazione incide sempre meno e la soluzione dei minimi quadrati tende a dare risultati diversi da 0.

```
betas    <- lasso$beta
df       <- lasso$df
MSE      <- lasso$RSS/n
bic      <- log(n)*df+n*log(MSE)
```

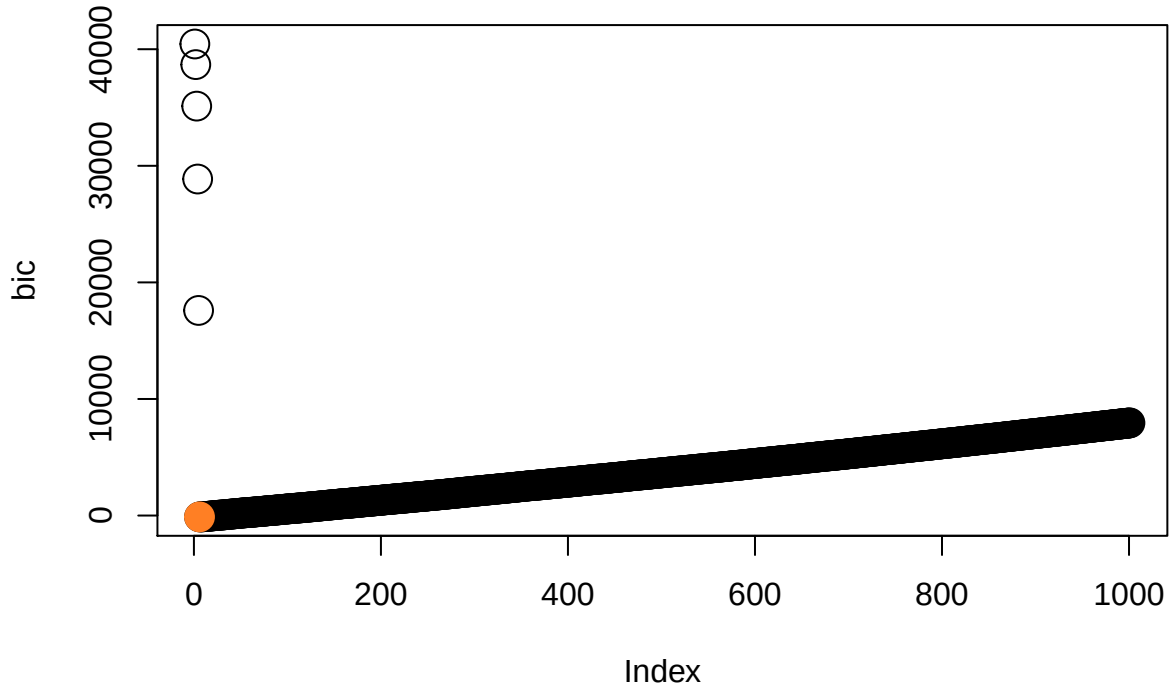
Con questi comandi siamo andati a ricavare dentro la struttura lars tutti gli elementi che ci servono per il calcolo del bic.

```
bestb <- which.min(bic)
bestb
```

```
##
## 6
```

Il bic ci suggerisce che il miglior modello è quello costituito da 6 parametri. Questo è il risultato che ci aspettavamo perchè oltre l'intercetta sappiamo che solo i primi 5 valori reali di beta sono diversi da 0. Di seguito riportiamo il grafico che rappresenta questa situazione.

```
plot(bic,cex=2)
points(bestb,bic[bestb],pch=19,cex=2, col="chocolate1")
```



Ricaviamo i valori dei coefficienti beta corrispondenti al modello con il miglior bic e mostraimo i primi 10.

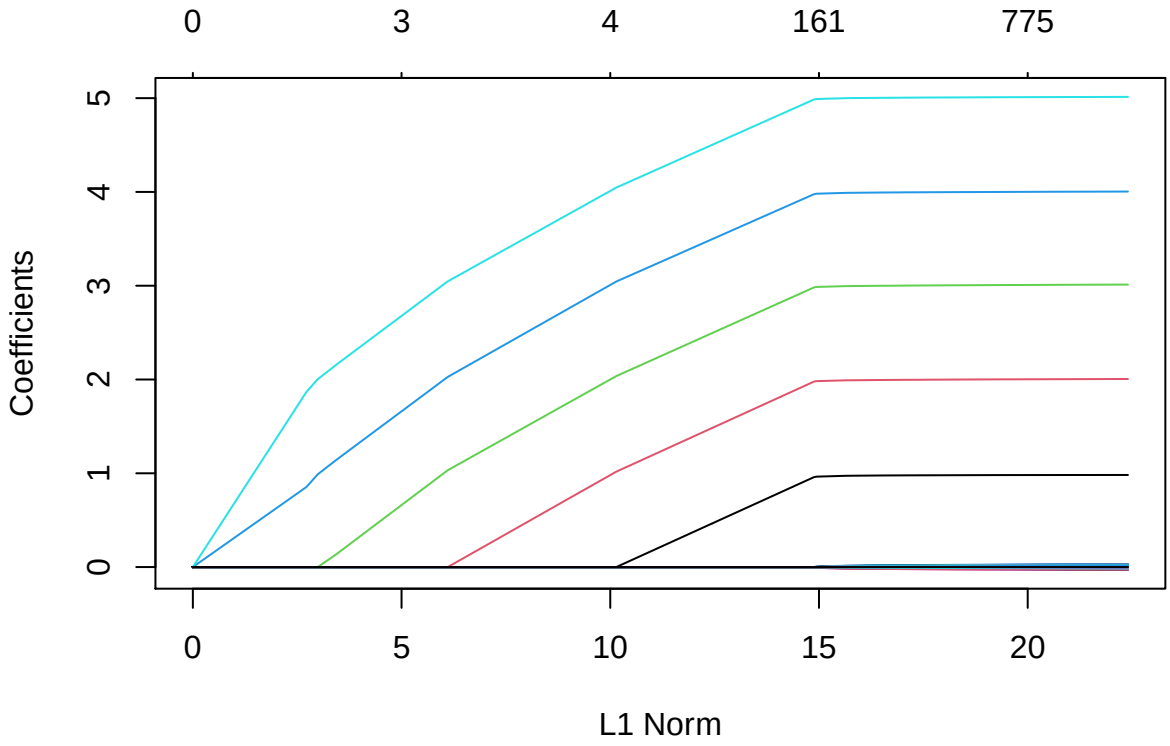
```
beta_lasso <- betas[bestb,]
kable(head(beta_lasso, n = 10), col.names = "Coefficients")
```

Coefficients
0.9554034
1.9740882
2.9782366
3.9725370
4.9821895
0.0000000
0.0000000
0.0000000
0.0000000
0.0000000

Glmnet

Carichiamo la libreria glmnet: per il calcolo della lasso impostiamo il valore di alpha uguale ad uno, e passiamo gli stessi valori di lambda usati nel calcolo effettuato con la lars.

```
library(glmnet)
glmnet <- glmnet(X, Y, alpha = 1, lambda = lasso$lambda, standardize=FALSE)
plot(glmnet)
```

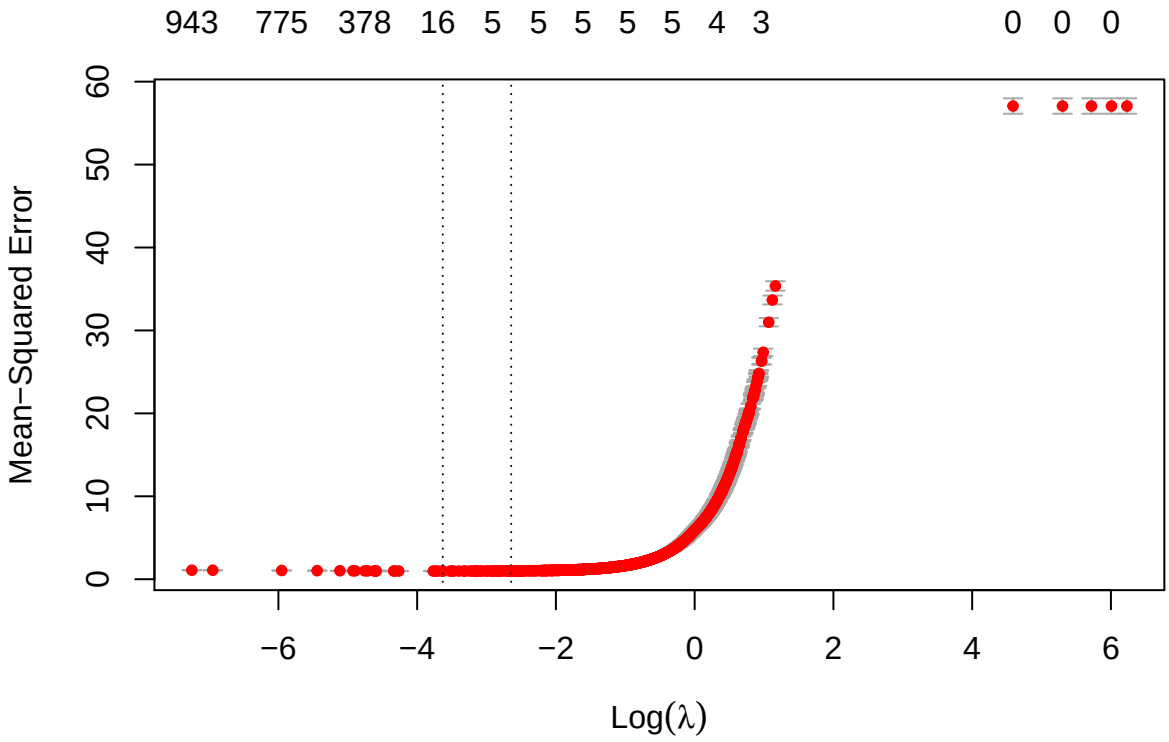


Ciascuna curva visualizza il valore di ciascun parametro al variare del fattore di penalizzazione e la norma L1-norm dell'intero vettore di coefficienti ottenuta al variare del fattore λ. Sopra è indicato il numero di coefficienti non nulli al variare di λ che sono denominati gradi di libertà effettivi (df) per il LASSO. A questo punto effettuiamo un'operazione di cross-validation per ricavare il miglior valore di lambda.

```
cv <- cv.glmnet(X,Y, alpha = 1, lambda = lasso$lambda)
best_lambda = cv$lambda.min
best_lambda
```

```
## [1] 0.02649772
```

```
plot(cv)
```



Nel grafico precedente il miglior valore di lambda si trova in corrispondenza della prima linea verticale. Da notare che sull'asse delle ascisse abbiamo il valore del $\log(\lambda)$.

Calcoliamo di nuova la regressione lasso tramite la funzione glmnet, passando come parametro il valore di lambda ottenuto con la cross-validation e mostriamo i primi 10 coefficienti

```
glmnet_best_lambda=glmnet(X, Y, alpha = 1, lambda = best_lambda , standardize=FALSE)
beta_glmnet_best_lambda=coef(glmnet_best_lambda)[-1]
kable(head(beta_glmnet_best_lambda,n=10 ),col.names = "Coefficients")
```

Coefficients
0.9609382
1.9796000
2.9837722
3.9780075
4.9875622
0.0000000
0.0000000
0.0000000
0.0000000
0.0000000

Infine confontiamo i risultati dei coefficienti beta ottenuti tramite OLS, lars e glmnet.

```
kable(cbind(coef_ols, coef_lasso, coef_glmnet_best_lambda))
```

	coef_ols	coef_lasso	coef_glmnet_best_lambda
X1	0.9818859	0.9554034	0.9609382
X2	2.0061491	1.9740882	1.9796000
X3	3.0127981	2.9782366	2.9837722
X4	4.0048383	3.9725370	3.9780075
X5	5.0142874	4.9821895	4.9875622
X6	-0.0080581	0.0000000	0.0000000
X7	0.0048219	0.0000000	0.0000000
X8	-0.0010329	0.0000000	0.0000000
X9	-0.0038628	0.0000000	0.0000000
X10	0.0178875	0.0000000	0.0000000

Come possiamo vedere con entrambe le funzioni quando il modello viene stiamto con il metodo LASSO i coefficienti non significativi vengono abbattuti a 0.

Simuliamo nuovamenete il nostro modello di regressione questa volta incrementando la deviazione standard della distribuzione da cui viene generato il termine di errore.

```
e_sd2=rnorm(n, sd=2)
Y_sd2 <- X%*%beta+e_sd2
Y_sd2 <- Y_sd2-mean(Y_sd2)
```

I risultati ottenuti con questo nuovo termine di errore sono i seguenti:

```
kable(cbind(coef_ols_sd2, coef_lasso_sd2, coef_glmnet_best_lambda_sd2))
```

	coef_ols_sd2	coef_lasso_sd2	coef_glmnet_best_lambda_sd2
X1	0.9944018	0.9266351	0.9428549
X2	1.9721883	1.9017867	1.9615207
X3	3.0095911	2.9450369	2.9658661
X4	4.0406414	3.9787541	3.9603470
X5	4.9687228	4.9099504	4.9698970
X6	0.0005420	0.0000000	0.0000000
X7	-0.0287737	0.0000000	0.0000000
X8	0.0099725	0.0000000	0.0000000
X9	0.0660520	0.0000000	0.0000000
X10	0.0076510	0.0000000	0.0000000

Mettiamo a confronto i valori ottenuti con i due diversi termini di errore con i differenti metodi di stima.

```
kable(cbind(coef_ols,coef_ols_sd2))
```

	coef_ols	coef_ols_sd2
X1	0.9818859	0.9944018
X2	2.0061491	1.9721883
X3	3.0127981	3.0095911

	coef_ols	coef_ols_sd2
X4	4.0048383	4.0406414
X5	5.0142874	4.9687228
X6	-0.0080581	0.0005420
X7	0.0048219	-0.0287737
X8	-0.0010329	0.0099725
X9	-0.0038628	0.0660520
X10	0.0178875	0.0076510

```
kable(cbind(coef_lasso,coef_lasso_sd2))
```

coef_lasso	coef_lasso_sd2
0.9554034	0.9266351
1.9740882	1.9017867
2.9782366	2.9450369
3.9725370	3.9787541
4.9821895	4.9099504
0.0000000	0.0000000
0.0000000	0.0000000
0.0000000	0.0000000
0.0000000	0.0000000
0.0000000	0.0000000

```
kable(cbind(coef_glmnet_best_lambda, coef_glmnet_best_lambda_sd2))
```

coef_glmnet_best_lambda	coef_glmnet_best_lambda_sd2
0.9609382	0.9428549
1.9796000	1.9615207
2.9837722	2.9658661
3.9780075	3.9603470
4.9875622	4.9698970
0.0000000	0.0000000
0.0000000	0.0000000
0.0000000	0.0000000
0.0000000	0.0000000
0.0000000	0.0000000

Con l'incremento della deviazione standard del termine di errore i risultati sono peggiorati in quanto i coefficienti stiamti sono più distanti dai coefficienti reali per tutti i metodi di stima utilizzati.

Confrontiamo infine i valori del MSE ottenuti.

```
MSE_ols <- mean((beta-beta_ols)^2)

MSE_lasso <- mean((beta-beta_lasso)^2)

MSE_glmnet <- mean((beta-beta_glmnet_best_lambda)^2)
```

```r
MSE_ols2 <- mean((beta-beta_ols_sd2)^2)

MSE_lasso2 <- mean((beta-beta_lasso_sd2)^2)
MSE_glmnet2 <- mean((beta-beta_glmnet_best_lambda_sd2)^2)

kable(rbind(MSE_ols, MSE_lasso, MSE_glmnet, MSE_ols2, MSE_lasso2, MSE_glmnet2),
      col.names = "Mean Squared Error")
```

	Mean Squared Error
MSE_ols	0.0001074
MSE_lasso	0.0000042
MSE_glmnet	0.0000029
MSE_ols2	0.0003805
MSE_lasso2	0.0000266
MSE_glmnet2	0.0000084

L'analisi di questi risultati conferma quanto affermato sopra: i mean squared error dei modelli generati con un incremento della deviazione standard della distibuzione dalla quale è stato estratto il termine di errore sono più alti. Inoltre la tabella ci mostra come nel nostro caso il risultato migliore è stato stimato con la funzione glmnet alla quale era stata assegnato il valore di λ ottenuto dalla cross_validation.

PROBLEMA 3

Introduzione al problema

Di seguito utilizzeremo la simulazione con l'obbiettivo di implementare l'algoritmo **Shooting**, ossia un'approccio per pervenire alla stima dei parametri di un modello di regressione regolarizzato mediante la procedura **LASSO**. I dati di simulazione verranno generati seguendo lo stesso approccio descritto nel corso del **Problema 2**, ma ponendo n=6 e p=3. L'analisi continua col proporre i risultati della regressione **OLS** e della regressione **Ridge**, i quali parametri stimati verranno impiegati per inizializzare l'algoritmo. Fissato poi un dato λ, ed ulteriori parametri necessari per regolare la condione di convergenza dell'algoritmo, si procede con l'implementazione dello stesso. L'analisi si conclude con un commento dei risultati ottenuti.

L'algoritmo Shooting

La soluzione LASSO per la stima dei parametri del modello di regressione basata sulla procedura **Shooting** è una soluzione **iterativa**: la procedura porta a stimare **uno alla volta** tutti i parametri del modello. Essa dipenderà in primo luogo dal valore del parametro di Tuning λ scelto, il quale può assumere valori compresi fra 0 ed ∞. Fissato un dato λ si procede con l'implementazione dell'algoritmo, che prevede l'esecuzione dei seguenti **passi**, per ogni iterazione t:

a) si fissa un regressore x_k, nonchè il corrispondente parametro β_k associato,
b) si procede con il calcolo delle seguenti componenti:

$$a_k = 2\sum_i x_{ik}^2$$

$$C_k = 2\sum_i (Y_i - x'_{i,-k}\beta_{-k})x_{ik}$$

c) Si confronta la componente C_k con il parametro di Tuning λ; tale confronto porta alla determinazione della stima di β_k. In particolare:

- se C_k<-λ, allora
$$\beta_k = (C_k + \lambda)/a_k$$
- se C_k>λ, allora
$$\beta_k = (C_k - \lambda)/a_k$$
- se C_k è compreso fra -λ e +λ allora, β_k=0

d) si ripetono i passi **a**, **b** e **c** per ciascuno dei restanti k-1 parametri,

e) il nuovo vettore dei parametri stimati, ottenuto nell'iterazione (t), viene confrontato con il vetore dei parametri stimati nell'iterazione precedente (t-1): se il vettore dei parametri stimato nell'iterazione t non si discosta molto dal vettore dei parametri stimato nell'iterazione t-1 l'agoritmo termina la sua esecuzione, altrimenti verrà effettuata un'ulteriore iterazione (si riparte, quindi, dal passo **a**).

Alcune considerazioni sulle componenti a_k e C_K

a_k rappresenta la somma dei valori relativi k-esimo regressore elevati al quadrato (per ogni osservazione i), e pertanto si caratterizza per essere una costante positiva. C_K rappresenta invece la somma dei residui del modello di regressione OLS calcolata rispetto ai restanti k-1 regressori/parametri. Notare che, per il calcolo di tale componente, occorre **inizializzare** il vettore dei coefficienti β. Tale vettore può essere inizializzato mediante una regressione Ridge, oppure mediante valori casuali; viene invece sconsigliato l'utilizzo della regressione OLS, specialmente nel caso in cui la soluzione LASSO viene impiegata per risolvere un problema di forte **multicollinearità**.

Generazione dei dati di simulazione

Come primo passo procediamo con il caricare le librerie che verranno impiegate durante l'analisi:

```
library(glmnet)
library(knitr)
```

Una volta caricate le librerie impostiamo il **seed**, ovvero il seme casuale da cui vengono generati i valori pseudocasuali: in questo modo i risultati ottenuti saranno replicabili.

```
set.seed(2015790003)
```

Per simulare l'algoritmo generiamo una matrice quadrata H di dimensioni (n x n), dove $n = 6$; tale matrice costituirà il Dataset di simulazione. I valori contenuti in H sono stati generati mediante il ciclo FOR descritto nel corso del **Problema 2** :

```
n = 6
H = matrix(nrow = n, ncol = n)
```

Per simulare il modello di regressione imponiamo dei valori ai coefficienti β_i. In particolare, nelle seguenti righe di codice generiamo il vettore dei coefficienti reali β di lunghezza $p = 3$, ponendo solo i primi due coefficienti diversi da 0:

```
p = 3
beta      <- rep(0,p)
beta[1:2] <- 1:2     # solo i primi 2 coef. sono diversi da zero
```

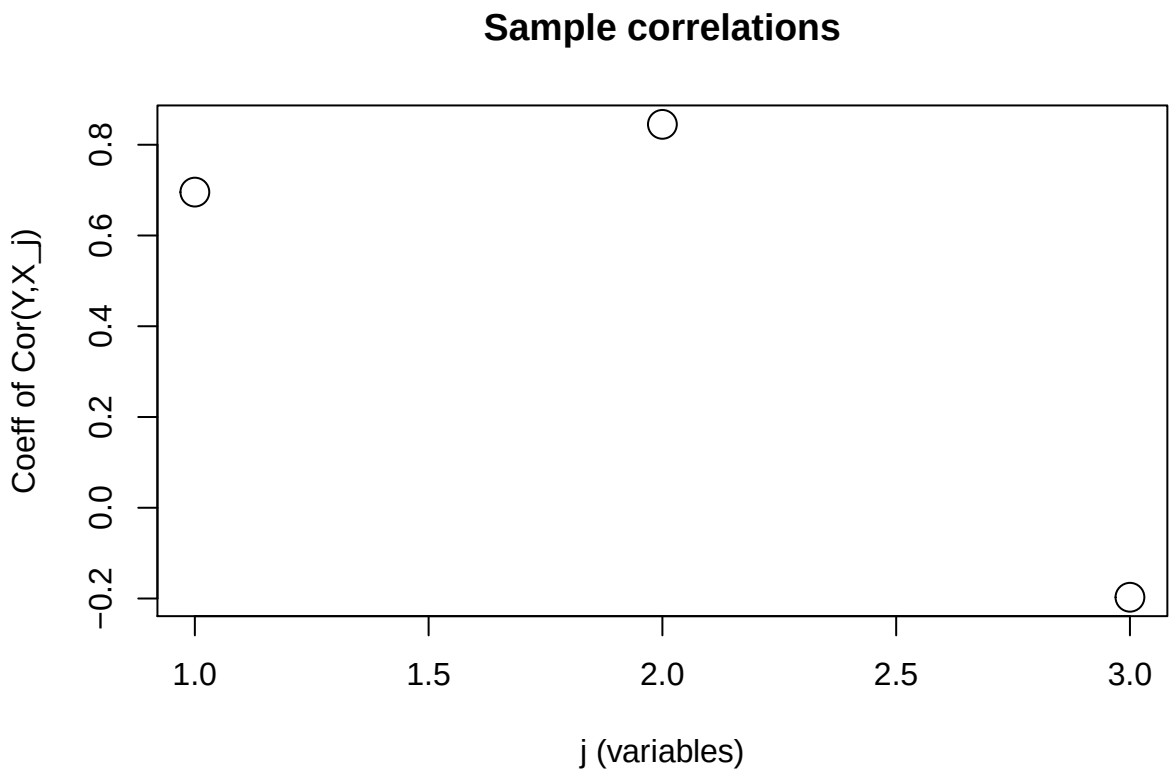
Dal Dataset di simulazione generiamo la matrice dei regressori X1 di dimensione (n x p), ponendo n=6 e p=3. Vengono quindi considerate solo le prime 3 colonne della matrice H.

```
X1= matrix(rnorm(H[,1:p]), nrow = n, ncol = p)  # tutte le righe e le prime 6 colonne di H
X1 <- scale(X1)
```

Dopo aver standardizzato la matrice X1, simuliamo il modello di regressione per generare il vettore (colonna) della variabile di risposta, che viene indicata con Y1.

Alla variabile di risposta Y1 viene aggiunto un termine di errore i cui valori sono generati da una distibuzione normale con media = 0 e deviazione standard = 1. Ai valori della variabile Y1 viene inoltre sottratta la sua media.

A questo punto andiamo a valutare l'associazione esistente tra la variabile di risposta Y1 ed i predittori contenuti nella matrice X1 mediante uno Scatterplot in cui vengono rappresentate graficamente le correlazioni tra la variabile di risposta Y1 ed i singoli predittori x_k:



Sull'asse delle ascisse vengono riportati i predittori x_k, mentre sull'asse delle ordinate vengono riportati i coefficienti di correlazione fra ciascun predittore e la variabile di risposta Y1. Dato che, solamente i primi due coefficienti sono stati posti diversi da zero, le correlazioni più significative le troviamo in corrispondenza dei primi due predittori. Per quanto riguarda il primo predittore x_1, il valore del coefficiente β_1 ad esso associato è stato posto inferiore rispetto a quello del secondo predittore x_2, e pertanto presenta una minore correlazione con Y1. Ciò sta a significare che all'aumentare del valore dei coefficienti β_i aumenteranno anche i valori delle correlazioni con la variabile di risposta Y1. Per quanto riguarda l'utimo predittore x_3 possiamo osservare che la correlazione con Y1 risulta essere bassa e negativa.

La regressione OLS

Usiamo il metodo dei minimi quadrati ordinari (OLS) per stimare i parametri del modello di regressione, e mostriamo i risultati ottenuti mediante la funzione summary():

```
regOLS <- lm(Y1~X1)
summary(regOLS)
```

```
##
## Call:
## lm(formula = Y1 ~ X1)
##
## Residuals:
##         1         2         3         4         5         6
##  0.059385 -0.004745 -0.057060  0.044509 -0.055538  0.013450
##
## Coefficients:
##                           Estimate            Std. Error t value Pr(>|t|)
## (Intercept) 0.0000000000000000541 0.0316903946577001411    0.00 1.000000
## X11         1.6129815400488645150 0.0478584207864735448   33.70 0.000879 ***
## X12         1.4895809804278787869 0.0366079735459531597   40.69 0.000603 ***
## X13         0.7883642483922430610 0.0459675725539734903   17.15 0.003383 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.07763 on 2 degrees of freedom
## Multiple R-squared:  0.9995, Adjusted R-squared:  0.9988
## F-statistic:  1370 on 3 and 2 DF,  p-value: 0.0007294
```

I risultati ottenuti evidenziano il fatto che la soluzione a minimi quadrati OLS non riesce a cogliere a pieno la realtà simulata. Dall'analisi dei p-value risulta che tutti i coefficienti stimati sono da considerarsi statisticamente significativi in quanto i p-value ad essi associati sono tutti minori di 0.05; ciò però non rispecchia la realtà simulata, in quanto il parametro β_3 è stato posto pari a zero. Inotre, confrontando i valori dei coefficienti stimati con i parametri reali simulati, si può notare che il metodo dei minimi quadrati ordinari tende a sovrastimare alcuni coefficienti di regressione:

	coefficienti simulati	coefficienti OLS
beta_1	1	1.6129815
beta_2	2	1.4895810
beta_3	0	0.7883642

Queste problematiche possono dipendere da vari fattori, quali ad esempio la **numerosità campionaria** (all'aumentare della numerosità del campione le stime tendono a migliorare) ed il problema della **multicollinearità**. Per meglio indagare le cause del problema analizziamo la matrice di correlazione tra i predittori x_k, in modo tale da verificare se esiste o meno multicollinearità tra i regressori:

```
cor(X1)
```

```
##            [,1]       [,2]       [,3]
## [1,]  1.0000000  0.3010479 -0.6505753
## [2,]  0.3010479  1.0000000 -0.1194922
## [3,] -0.6505753 -0.1194922  1.0000000
```

La matrice di correlazione permette di cogliere il grado di associazione lineare fra le variabili considerate: se i predittori tendono a fornire informazioni simili la loro correlazione tenderà ad assumere valori elevati. Tenuto conto che tale indice assume valori compresi tra -1 ed 1, possiamo concludere che la correlazione maggiormente significativa viene riscontrata tra il predittore x_1 ed il predittore x_3. Tale correlazione è negativa e pari a -0.65.

La regressione Ridge

Precedentemente è stato affermato che, per il calcolo della componente C_k occore inizializzare il vettore dei coefficienti β. Tale vettore può essere inizializzato ricorrendo ad una regressione Ridge, oppure mediante l'assegnazione di valori casuali. In questa analisi decidiamo di inizializzare il vettore dei parametri β ricorrendo ad una regressione Ridge. Il motivo di tale decisione è dovuto al fatto che, mediante tale approccio, ci aspettiamo di giungere più rapidamente a convergenza, in quanto la regressione ridge ci permette di partire da una soluzione già parzialmente regolarizzata.

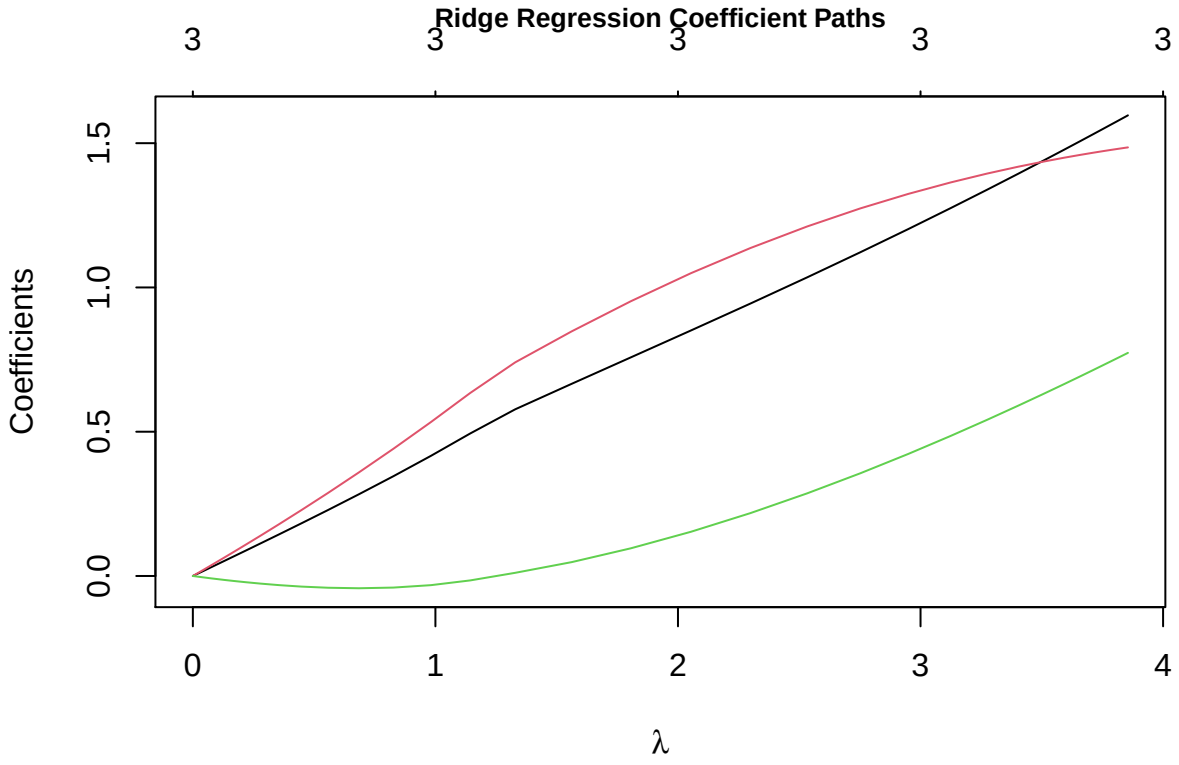
Procediamo quindi con la stima dei parametri del modello mediante la regressione Ridge e, come primo passo, fissiamo una griglia di valori relativi al parametro di Tuning λ.

```
lambdaGrid <-10^seq(10, -2, length = 100) # proposte dei valori di lambda
```

Eseguiamo poi il Fit del modello di regressione Ridge utilizzando la griglia di valori relativi al parametro di Tuning λ. La funzione glmnet() permette di calcolare tutte le soluzioni relative alla regressione Ridge, per tutti i valori di λ contenuti all'interno della griglia. I parametri passati in input alla funzione consistono nella matrice dei regressori X1, il vettore relativo alla variabile di risposta Y1 e la griglia di valori di λ (lamdaGrid). Con alpha= 0 specifichiamo alla funzione di eseguire una regressione Ridge.

```
ridgeReg <- glmnet(X1, Y1, alpha = 0, lambda = lambdaGrid)
```

La rappresentazione grafica che segue mostra il trend dei coefficienti stimati al variare di λ:



Sull'asse delle ascisse viene riportato il reciproco di λ, ossia $1/\lambda$, mentre sull'asse delle ordinate vengono riportati i valori dei coefficienti stimati per ogni dato λ. Dalla rappresentazione grafica emerge che, man mano che ci spostiamo verso destra sull'asse delle ascisce (e quindi verso una soluzione OLS) il valore dei coefficienti tende ad aumentare in maniera incontrollata.

Per scegliere il valore del parametro di Tuning λ da impiegare per inizializzare il vettore dei coefficienti β, nell'ambito dell'algoritmo Shooting, ci avvaliamo della procedura di **cross-validation**. Utilizziamo quindi la funzione di CV incorporata nella funzione glmnet(), ovvero la funzione cv.glmnet(): tale funzione esegue una scelta automatizzata di λ sulla base del valore minimo di MSE. Nello specifico, vengono considerati tutti i possibili valori di λ passati in input alla funzione, la quale sceglierà il λ che minimizza il MSE ottenuto mediante una operazione di cross-validazione.

```
cv_ridgeReg <- cv.glmnet(X1,Y1, alpha = 0 , lambda = lambdaGrid, grouped = FALSE )
```

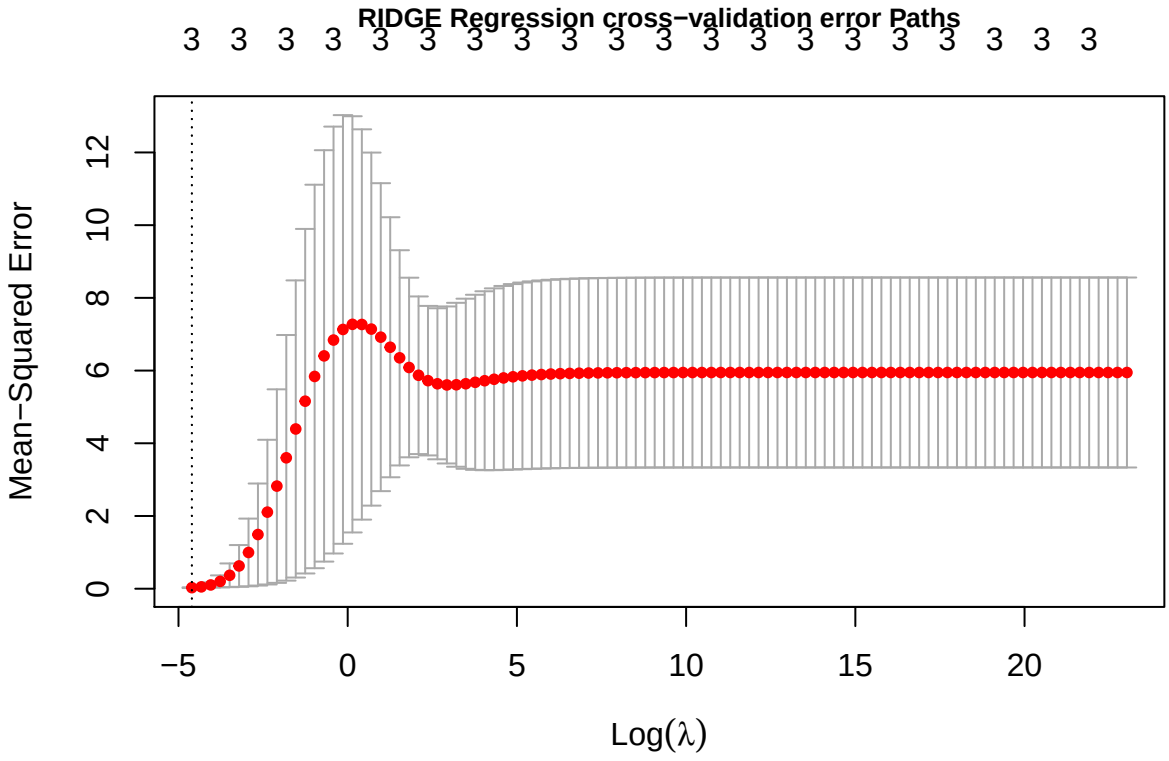
Con i seguenti comandi selezioniamo e visualizziamo il valore di λ che minimizza il MSE calcolato mediante la procedura di cross-validation:

```
best_lambda <- cv_ridgeReg$lambda.min
best_lambda
```

```
## [1] 0.01
```

Abbiamo quindi ottenuto il valore di λ che ci permette di ottenere il più piccolo errore di cross-validazione. La rappresentazione grafica che segue mostra l'andamento del Mean Square Error di CV in funzione di λ:

```
plot(cv_ridgeReg , main = list( "RIDGE Regression cross-validation error Paths" , cex = 0.8 ) )
```




```
log(best_lambda) # mostra il logaritmo di best_lambda
```

```
## [1] -4.60517
```

Il valore del parametro di Tuning λ, selezionato mediante la procedura di CV, verrà ora impiegato per inizializzare il vettore dei coefficienti β necessario per poter calcolare la componente C_k nell'ambito dell'algoritmo Shooting. Rieseguiamo quindi la regressione ridge utilizzando il valore di λ ottenuto mediante procedura di CV e selezioniamo i coefficienti stimati:

```
ridgeReg_TOP <- glmnet(X1 , Y1 , alpha = 0 , lambda = best_lambda)
coef_TOP <- ridgeReg_TOP$beta #vettore dei coefficienti ridge
coef_TOP
```

```
## 3 x 1 sparse Matrix of class "dgCMatrix"
##           s0
## V1 1.5965151
## V2 1.4854187
## V3 0.7733488
```

l'oggetto **coef_TOP** è un vettore contenente i coefficienti stimati mpiegando il λ ottenuto mediante la procedura di CV.

L'implementazione dell'algoritmo Shooting

Procediamo ora con l'implementazione dell'algoritmo Shooting, per poter così pervenire alla stima dei parametri del modello di regressione mediante la procedura di regolarizzazione **LASSO**. Nelle righe di codice che seguono vengono inizializzati i parametri necessari per poter implementare l'algoritmo:

```
coef_Killing_t <- coef_TOP
coef_Killing_t=as.vector(coef_Killing_t) # vettore dei coefficienti stimati durante
                                         #l'iterazione corrente (t)
coef_iniziali=coef_Killing_t

lambdaK <- 2.9232                        # valore di lambda fissato
valueConv <- 0.0005                      # treshold

fermati <- TRUE                          # variabile booleana: regola il numero di iterazioni
count=0                                  # conta il numero di iterazioni eseguite dall'algoritmo
coef_Killing_tp <- coef_Killing_t        # vettore dei coefficienti stimati nell'iterazione precedente
                                         #(t-1)
label0 <- "vettore dei coefficienti Ridge iniziale: "
label1 <- "iterazione: "
label2 <- "coefficienti stimati: "
```

In particolare, sono stati creati tre vettori:

- il vettore **coef__Killing__t** conterrà i coefficienti LASSO stimati durante l'iterazione t
- il vettore **coef__Killing__tp** conterrà i coefficienti LASSO stimati durante l'iterazione t-1
- il vettore **coef__iniziali** conterrà i coefficienti impiegati per inizializzare l'algoritmo, ossia, i coefficienti di regressione Ridge stimati mediante il valore λ selezionato mediante la procedura di cross-validation.

Viene ioltre fissato un λ pari a 2.9232 ed un valore di **threshold** pari a 0.0005; quest'ultimo parametro verrà impiegato per imporre la condizione di convergenza, e pertanto può essere considerato come una condizione di termine dell'algoritmo. Al parametro di threshold viene inoltre associato un ulteriore parametro di tipo booleano, denominato **"fermati"**: quando tale parametro assumerà valore **FALSE** l'algoritmo terminerà la sua esecuzione. Affinchè tale parametro assuma valore FALSE, è necessario che la differenza fra il valore assoluto del vettore dei coefficienti stimati nell'iterazione t (**coef__Killing__t**) ed il valore assoluto del vettore dei coefficienti stimati nell'iterazione t-1 (**coef__Killing__tp**) sia minore o uguale al parametro di threshold **"valueConv"**. La velocità (e quindi il numero di iterazioni) con cui l'algoritmo raggiunge la convergenza (arrestandosi) dipenderà quindi:

- dal vettore dei parametri di partenza,
- dal treshold fissato.

Per tener traccia del numero di iterazioni eseguite dall'algoritmo viene fissato un contatore (**count**), che si incrementerà ad ogni iterazione eseguita. Il parametro di Tuning λ determina invece il peso attribuito alla componente di regolarizzazione: valori di λ crescenti tendono a favorire una soluzione maggiormente sparsa (si attribuisce maggior peso alla componente di regolarizzazione LASSO). Pertanto, l'output generato dall'algoritmo dipenderà:

- dal vettore dei parametri β con cui inizializziamo la procedura (numero di iterazioni dell'algoritmo),
- dal parametro di Tuning λ fissato (peso attribuito alla componente di regolarizzazione),
- dal parametro di threshold fissato (numero di iterazioni dell'algoritmo).

Procediamo ora con l'implementazione e l'esecuzione dell'algoritmo:

```
while( fermati==TRUE ) {

  for (k in 1: length(coef_Killing_t)) {

    Xk <- X1[,k]                                  # vettore del k-esimo regressore
    Xk1 <- cbind(Xk)                              # incolonna i valori di Xk
                                                  # => lo uso per il calcolo di Ck
    XkRestanti <- X1[,-k]                         # matrice dei restanti k-1 regressori
    betaRestanti <- coef_Killing_t[-k]            # vettore dei restanti k-1 parametri

    ak <- 2*(sum((Xk)^2))                         # calcolo di ak

    residui <- Y1 - (XkRestanti%*%betaRestanti) # faccio Y - X*beta
                                                  # (per i rimanenti k-1 regressori/parametri)
    Ck <- 2*(sum(residui * Xk1))

    if(Ck < -lambdaK){                            # confronto Ck e lambda per stimare betaK

      coef_Killing_t[k] <- (Ck + lambdaK)/ak

    } else {

      if(Ck > lambdaK){

        coef_Killing_t[k] <- (Ck - lambdaK)/ak

      } else {

        coef_Killing_t[k] <- 0

        fermati=FALSE
        break

      }

   }

  }

  sumTp <- ( sum( abs(coef_Killing_tp ) ) ) # somma dei valori assoluti dei coefficienti
                                            # stimati nell'iterazione precedente

  sumT <- ( sum( abs(coef_Killing_t ) ) )   # somma dei valori assoluti dei coefficienti
                                            # stimati nell'iterazione corrente

  if(count==0){
    print("*********************************************************************************"
          , quote = FALSE)
    print("l' output che segue mostra i risultati ottenuti durante ciascuna iterazione",
          quote = FALSE)
    print("*********************************************************************************"
          , quote = FALSE)
```

```
    print(" " , quote = FALSE)
    print( label0 , quote = FALSE )
    print(" " , quote = FALSE)
    print( c(" " , coef_iniziali ) , quote = FALSE ) # stampo il vettore dei
                                                     # coefficienti di partenza (Ridge)
    print(" " , quote =FALSE)

  }

  if( abs(sumT-sumTp) <= valueConv ){           # verifico la condizione di convergenza

      fermati <- FALSE
      count <- count+1
      print( c(label1 , count) , quote = FALSE )
      print(" " , quote =FALSE)
      print( c(label2 , coef_Killing_t) , quote = FALSE ) # stampa i coefficienti
                                                          # dell'iterazione corrente
      print(" " , quote =FALSE)
      print("*********************************************************************************",
            quote = FALSE)
      print(" " , quote = FALSE)

    }else{

      fermati <- TRUE
      coef_Killing_tp <- coef_Killing_t         # aggiorno il vettore coef_Killing_tp
                                                # con i valori dell'iterazione corrente
      count=count+1
      print( c(label1 , count) , quote = FALSE)
      print(" " , quote = FALSE)
      print( c(label2 , coef_Killing_tp ) , quote = FALSE) # stampa i coefficienti
                                                           # dell'iterazione corrente
      print(" " , quote = FALSE)
      print("*********************************************************************************",
            quote = FALSE)
      print(" " , quote = FALSE)
  }

}
```

```
## [1] ********************************************************************************
## [1] l' output che segue mostra i risultati ottenuti durante ciascuna iterazione
## [1] ********************************************************************************
## [1]
## [1] vettore dei coefficienti Ridge iniziale:
## [1]
## [1]                   1.59651510317108  1.48541870258759  0.773348796931953
## [1]
## [1] iterazione:  1
## [1]
## [1] coefficienti stimati:  1.31214590312324       1.28603269921507
## [4] 0.276005569738571
## [1]
```

```
## [1] ********************************************************************************
## [1]
## [1] iterazione:  2
## [1]
## [1] coefficienti stimati:  1.04861142093435       1.30594055888183
## [4] 0.106935375080687
## [1]
## [1] ********************************************************************************
## [1]
## [1] iterazione:  3
## [1]
## [1] coefficienti stimati:  0.932625305324019      1.32065536476792
## [4] 0.0332359761799226
## [1]
## [1] ********************************************************************************
## [1]
## [1] iterazione:  4
## [1]
## [1] coefficienti stimati:  0.880248433532369      1.32761680829373
## [4] 0
## [1]
## [1] ********************************************************************************
## [1]
## [1] iterazione:  5
## [1]
## [1] coefficienti stimati:  0.856530199565309      1.33078569272176
## [4] 0
## [1]
## [1] ********************************************************************************
## [1]
## [1] iterazione:  6
## [1]
## [1] coefficienti stimati:  0.855576213437746      1.33107288827977
## [4] 0
## [1]
## [1] ********************************************************************************
## [1]
## [1] iterazione:  7
## [1]
## [1] coefficienti stimati:  0.855489753806775      1.33109891677352
## [4] 0
## [1]
## [1] ********************************************************************************
## [1]
```

Analizzando l'output ottenuto osserviamo che, per un λ pari a 2.9232 ed un valore di threshold pari a 0.0005 l'algoritmo eseguirà sette iterazioni. Osservando invece i vettori dei coefficienti LASSO stimati in ciascuna iterazione, notiamo che tali valori tendono a rispecchiare i valori reali simulati. In particolare, nella quarta iterazione, il coefficiente β_3 relativo al regressore x_3 viene abbattuto e posto pari a zero, rispecchiando in questo modo la realtà simulata. Possiamo quindi concludere che la regressione LASSO:

- fornisce una stima **regolarizzata** dei coefficieti di regressione,
- allo stesso tempo, però, effettua anche una **model selection** , ossia indica quali sono i regressori che esercitano un impatto significativo sulla variabile di risposta; infatti, porre la stima di un dato

coefficiente β_k pari a zero implica abbattere l'effetto che il k-esimo regressore (x_k) esercita sulla variabile di risposta.

Nella tabella che segue mettiamo a confronto i risultati ottenuti durante l'analisi:

	coefficienti simulati	coefficienti OLS	coefficienti RIDGE	coefficienti LASSO
beta_1	1	1.6129815	1.5965151	0.8554898
beta_2	2	1.4895810	1.4854187	1.3310989
beta_3	0	0.7883642	0.7733488	0.0000000

Osservando la tabella emerge che la soluzione LASSO, ottenuta mediante l'implementazione dell'algoritmo Shooting, rappresenta la soluzione migliore, ossia, rappresenta la soluzione che ci permette di ottenere i coefficienti di regressione stimati che più si avvicinano ai veri valori dei coefficienti simulati.

Resources:

Statistical Learning program, L. Ippoliti
Sparse_Regressione_Hitter_data.R, L. Ippoliti, Statistical Learning program
Data Science program, P. Positiglione
https://www.lorenzogovoni.com/tre-tecniche-di-regolarizzazione-ridge-lasso-ed-elastic-net/
https://www.sciano.net/post/3/lasso-vs-ridge-regression
https://it.wikipedia.org/wiki/Regolarizzazione_(matematica)
https://www.pluralsight.com/guides/linear-lasso-and-ridge-regression-with-r
https://www.datacamp.com/community/tutorials/tutorial-ridge-lasso-elastic-net
https://www.statology.org/variance-inflation-factor-r/
https://it.wikipedia.org/wiki/Regressione_lineare#Multicollinearit%C3%A0