SLLD - Module 2

Ridge and Lasso

S.Tonini, F.Chiaromonte

Sant'Anna School of Advanced Study - Pisa

6/3/2025

Libraries

We are going to use

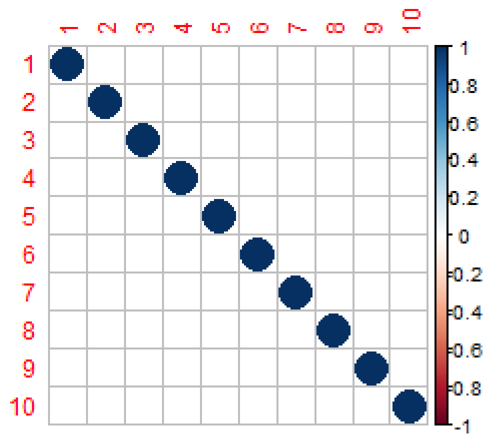
```
library(glmnet)     # ridge and lasso for GLMs
library(tidyverse)  # data manipulation and visualization
library(caret)      # statistical learning techniques
library(ggplot2)    # plots
library(corrplot)   # correlation matrix plotting
library(mvtnorm) # Sampling from a multivariate Normal
library(clusterGeneration) # Random matrix generation
```

Data

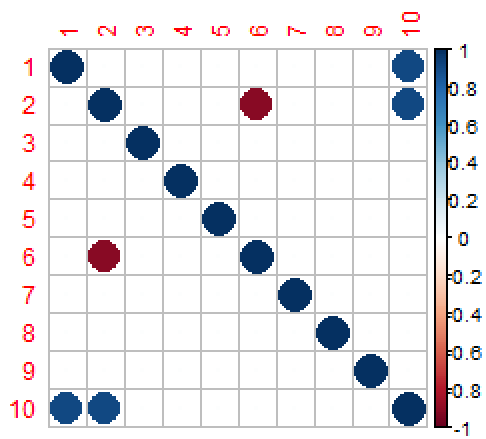
We simulate data as follows

```
p <- 10
set.seed(123)
Sigma1 = diag(p)
Sigma2 = diag(p)
Sigma2[10,1]=Sigma2[1,10]=0.9
Sigma2[10,2]=Sigma2[2,10]=0.9
Sigma2[6,2]=Sigma2[2,6]=-0.9
```

```
corrplot(Sigma1)
```




```
corrplot(Sigma2)
```



```
n<-100
set.seed(1)
x1 <- rmvnorm(n=n, replicate(p,0), Sigma1)
beta <- replicate(5,1)
y1 <- x1[,1:5]%*%beta + rnorm(n, 0, 0.3)
df1 <- cbind(y1,x1)
x2 <- rmvnorm(n, sigma=Sigma2)
y2 <- x2[,1:5]%*%beta + rnorm(n, 0, 0.3)
df2 <- cbind(y2,x2)
```

Penalized regression

We will perform ridge/lasso penalization through the **glmnet** package. The function **glmnet()** fits a generalized linear model via penalized maximum likelihood. The regularization path is computed for the lasso or elasticnet penalty at a grid of values for the regularization parameter lambda. The main arguments are:

- **x:** input matrix
- **y:** response variable
- α **:** the elastic-net mixing parameter with range [0, 1]. Namely, $\alpha = 1$ is the lasso (default) and $\alpha = 0$ is the ridge.
- **standardize:** a logical flag for x variable standardization, prior to fitting the model sequence. The coefficients are always returned on the original scale. Default is standardize=TRUE.

Ridge

To perform ridge regression, we run glmnet with α=0. The λ's sequence is internally computed by the package itself – although a user-defined sequence can be provided.

```
ridge <- glmnet(x1, y1, alpha=0)
summary(ridge)
```

```
##           Length Class     Mode
## a0         100   -none-    numeric
## beta      1000   dgCMatrix S4
## df         100   -none-    numeric
## dim          2   -none-    numeric
## lambda     100   -none-    numeric
## dev.ratio  100   -none-    numeric
## nulldev      1   -none-    numeric
## npasses      1   -none-    numeric
## jerr         1   -none-    numeric
## offset       1   -none-    logical
## call         4   -none-    call
## nobs         1   -none-    numeric
```

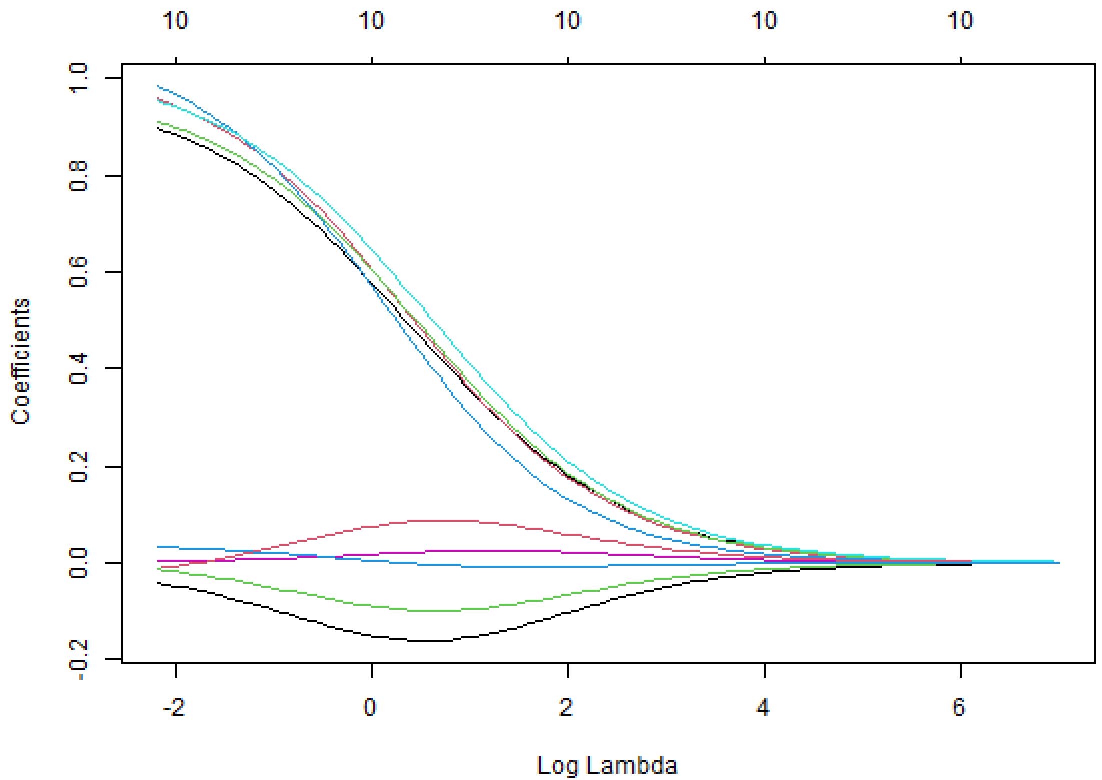
The summary is quite different than the one for linear regression, since ridge regression requires the tuning of λ. The code above fits a ridge regression for each λ value, and we have access to each of these model estimates.

We can plot the regularization path as follows:

```
dim(ridge$beta)
```

```
## [1]  10 100
```

```
plot(ridge, xvar="lambda")
```



We can automate the task of finding the optimal lambda value using the **cv.glmnet** function. This performs a k-fold cross-validation for **glmnet**, produces a plot, and returns "optimal' ' λ values.

```
cv_ridge <- cv.glmnet(x1, y1, alpha = 0)
cv_ridge
```

```
##
## Call:  cv.glmnet(x = x1, y = y1, alpha = 0)
##
## Measure: Mean-Squared Error
##
##     Lambda Index Measure      SE Nonzero
## min 0.1106   100  0.1244 0.01936      10
## 1se 0.1462    97  0.1394 0.02256      10
```

Two particular values of λ are highlighted: the minimum (min) and the largest value of lambda such that error is within 1 standard error of the minimum (1se).

```
cv_ridge$lambda.min
```

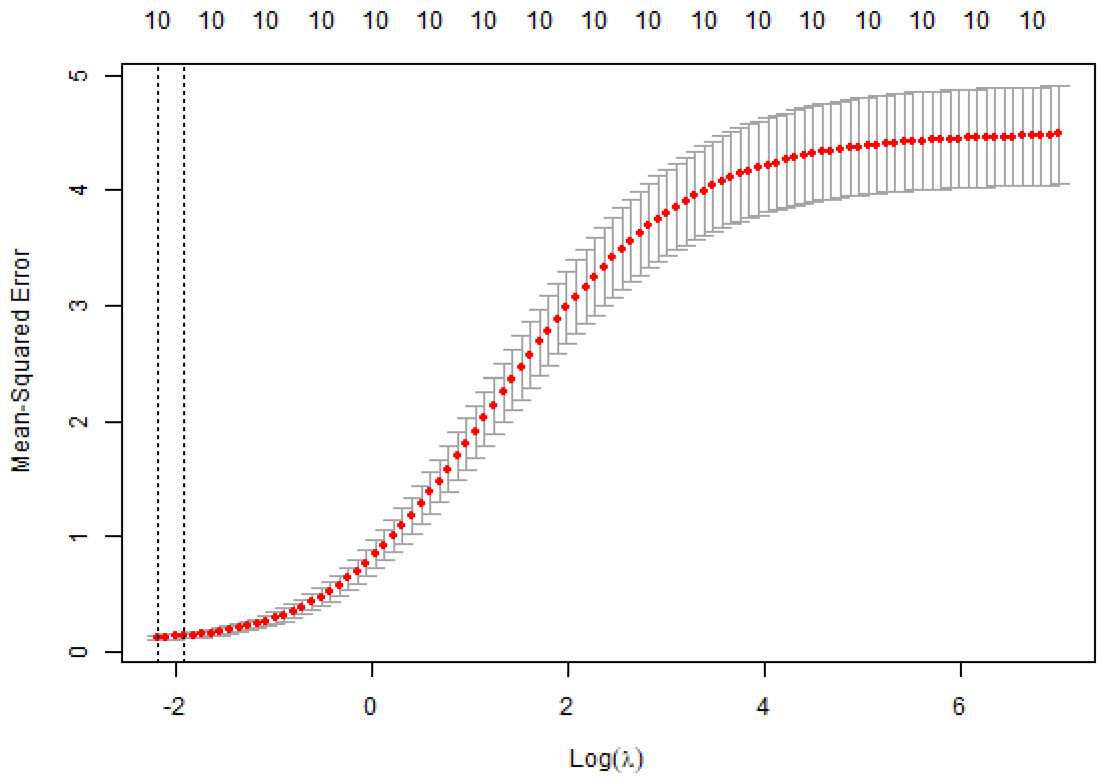
```
## [1] 0.1105733
```

```
cv_ridge$lambda.1se
```

```
## [1] 0.1461714
```

We can visualize them in this way:

```
plot(cv_ridge)
```

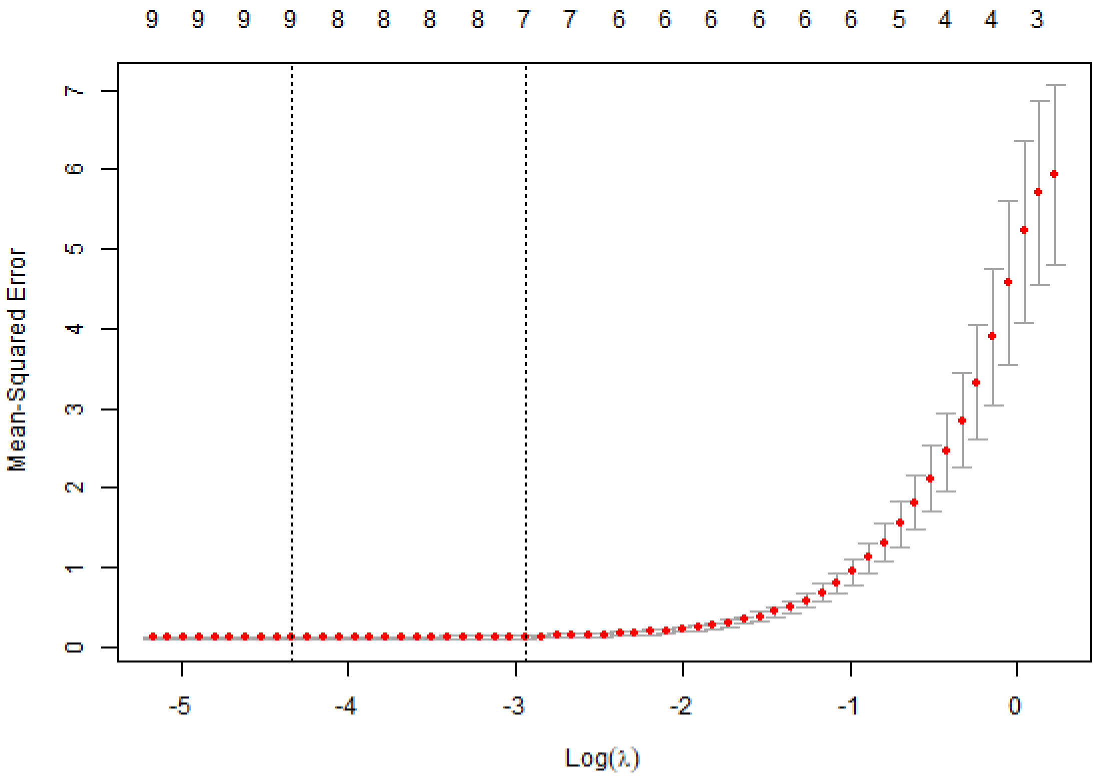


LASSO

Let us now perform Lasso regression using the glmnet package. We follow the same approach as in Ridge regression, but set $\alpha = 1$

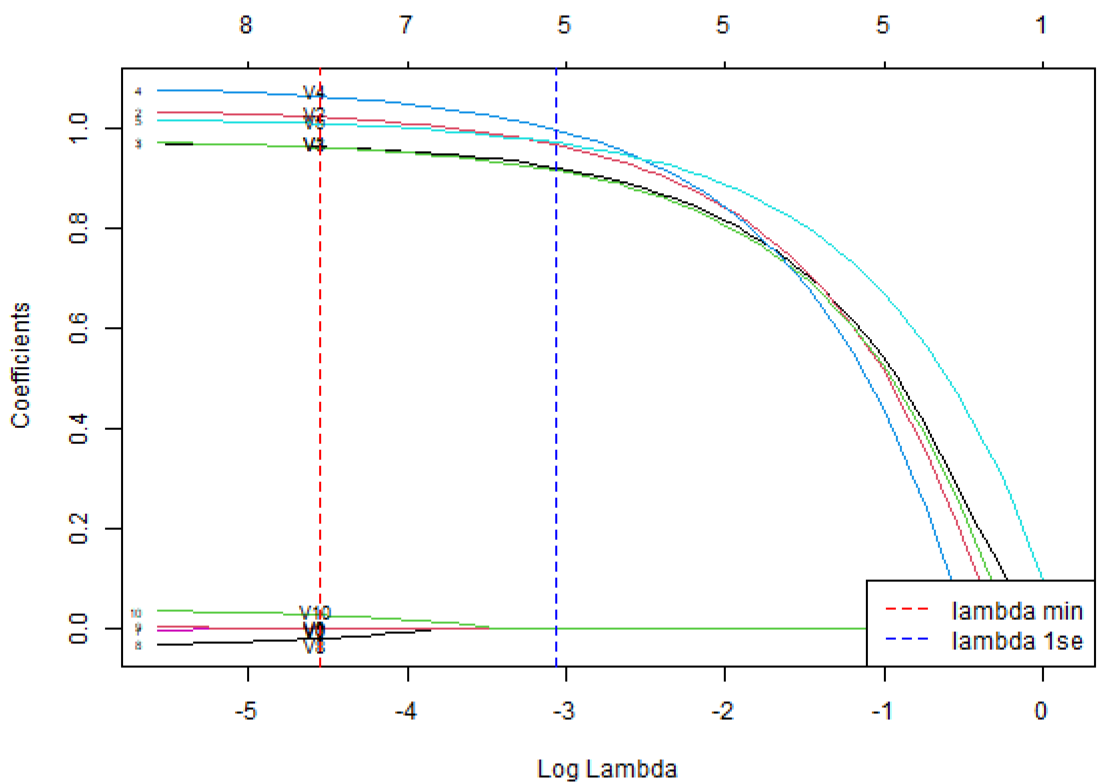
```
lasso1 <- glmnet(x1, y1, alpha=1)
cv_lasso1 <- cv.glmnet(x1, y1, alpha = 1)
lasso2 <- glmnet(x2, y2, alpha=1)
cv_lasso2 <- cv.glmnet(x2, y2, alpha = 1)

lbs_fun <- function(fit, offset_x=1, ...) {
L <- length(fit$lambda)
x <- log(fit$lambda[L])+ offset_x
y <- fit$beta[, L]
labs <- names(y)
text(x, y, labels=labs, cex=0.75, ...)
}
```

```
plot(cv_lasso2)
```

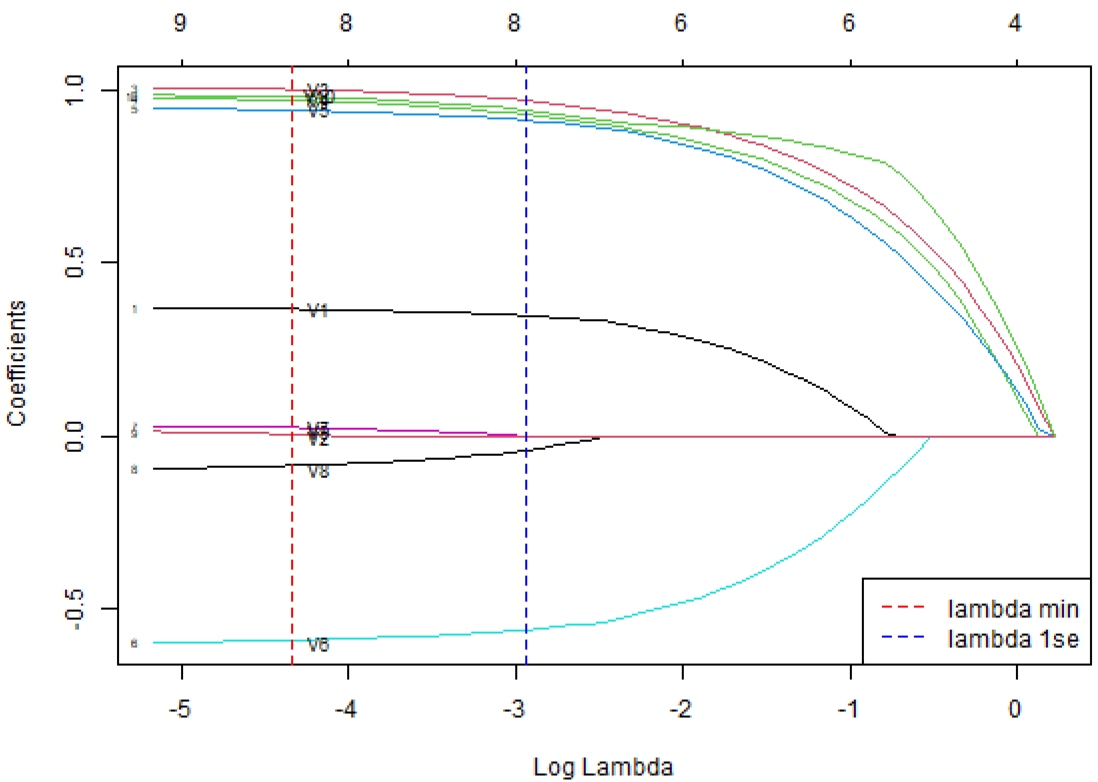


Let us see how the regression coefficients change by modifying λ:

```
plot(lasso1, xvar = "lambda", label=T)
lbs_fun(lasso1)
abline(v=log(cv_lasso1$lambda.min), col = "red", lty=2)
abline(v=log(cv_lasso1$lambda.1se), col="blue", lty=2)
legend(x = "bottomright", legend = c("lambda min", "lambda 1se"),
lty = c(2, 2), col = c("red", "blue"))
```



```r
plot(lasso2, xvar = "lambda", label=T)
lbs_fun(lasso2)
abline(v=log(cv_lasso2$lambda.min), col = "red", lty=2)
abline(v=log(cv_lasso2$lambda.1se), col="blue", lty=2)
legend(x = "bottomright",
legend = c("lambda min", "lambda 1se"),
lty = c(2, 2),
col = c("red", "blue"))
```



Ridge and LASSO with Information Criterion

```r
library(HDeconometrics)
```

We perform penalization through the **HDeconometrics** package. The function **ic.glmnet** allows us to estimate a GLM with lasso, elasticnet or ridge regularization using information criterion. It avoids some complications of cross-validation in time-series.

```r
lasso_BIC1 <- ic.glmnet(x1, y1, crit = "bic")
lasso_BIC1$lambda
```

```
## [1] 0.0322327
```

```r
lasso_BIC2 <- ic.glmnet(x2, y2, crit = "bic")
lasso_BIC2$lambda
```

```
## [1] 0.0174015
```

Let us rebuild the model and compare the estimated coefficients for min and 1se λ.

```
min_lasso1 <- glmnet(x1, y1, alpha=1, lambda= cv_lasso1$lambda.min)
se_lasso1 <- glmnet(x1, y1, alpha=1, lambda= cv_lasso1$lambda.1se)
min_lasso2 <- glmnet(x2, y2, alpha=1, lambda= cv_lasso2$lambda.min)
se_lasso2 <- glmnet(x2, y2, alpha=1, lambda= cv_lasso2$lambda.1se)
lasso_mat <- cbind(coef(min_lasso1), coef(se_lasso1),
                   lasso_BIC1$coefficients, coef(min_lasso2),
                   coef(se_lasso2), lasso_BIC2$coefficients)
colnames(lasso_mat) <- c("x1_min", "x1_1se", "x1_bic", "x2_min",
                         "x2_1se", "x2_bic")
lasso_mat[,1:3]
```

```
## 11 x 3 sparse Matrix of class "dgCMatrix"
##                    x1_min      x1_1se      x1_bic
## (Intercept)  0.0060195430 0.004081654 0.005787302
## V1           0.9616428042 0.919692520 0.937039701
## V2           1.0208922537 0.966049084 0.986742026
## V3           0.9605879276 0.914761272 0.932609192
## V4           1.0628414516 0.995947325 1.021470111
## V5           1.0079865828 0.969626939 0.983313205
## V6           .            .           .
## V7          -0.0001048491 .           .
## V8          -0.0209647223 .           .
## V9           .            .           .
## V10          0.0259341480 .           .
```

```r
lasso_mat[,4:6]
```

```
## 11 x 3 sparse Matrix of class "dgCMatrix"
##                   x2_min       x2_1se      x2_bic
## (Intercept)  0.012820004  0.005360225  0.01156280
## V1           0.366774187  0.347429030  0.36379493
## V2           .            .            .
## V3           1.004338791  0.972115997  1.00116840
## V4           0.972414637  0.931835248  0.96823245
## V5           0.943921984  0.915280215  0.94056787
## V6          -0.594149179 -0.564772826 -0.59029867
## V7           0.023743918  .            0.02090666
## V8          -0.086377928 -0.041366994 -0.08177566
## V9           0.003709776  .            .
## V10          0.982488433  0.943333881  0.97951439
```

Now it's your turn!!!