

# **SLLD - Module 2**

## **Feature Screening**

**S.Tonini, F.Chiaromonte**

**Sant'Anna School of Advanced Study - Pisa**

**14/3/2025**

## Libraries

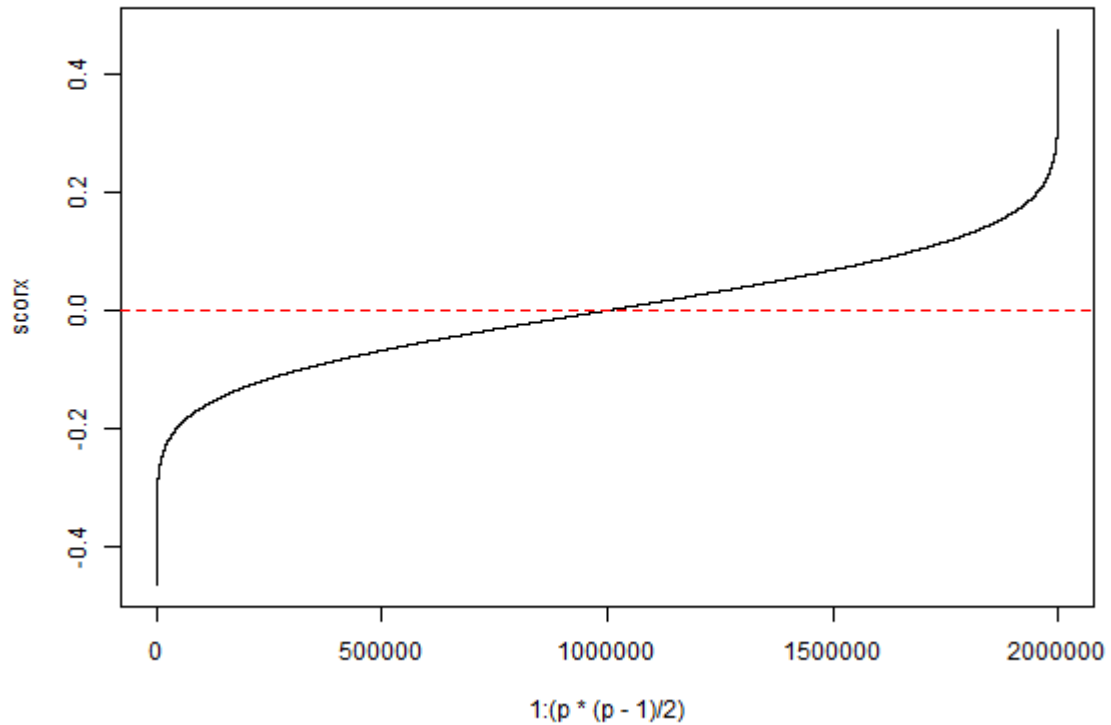
```
if(!require(SIS)){install.packages("SIS", dep=T); library(SIS)}  
library(glmnet) # elastic net for GLMs  
library(mvtnorm) # to generate multivariate normal distributions  
library(corrplot) # correlation plot
```

## Data: “Ideal” scenario

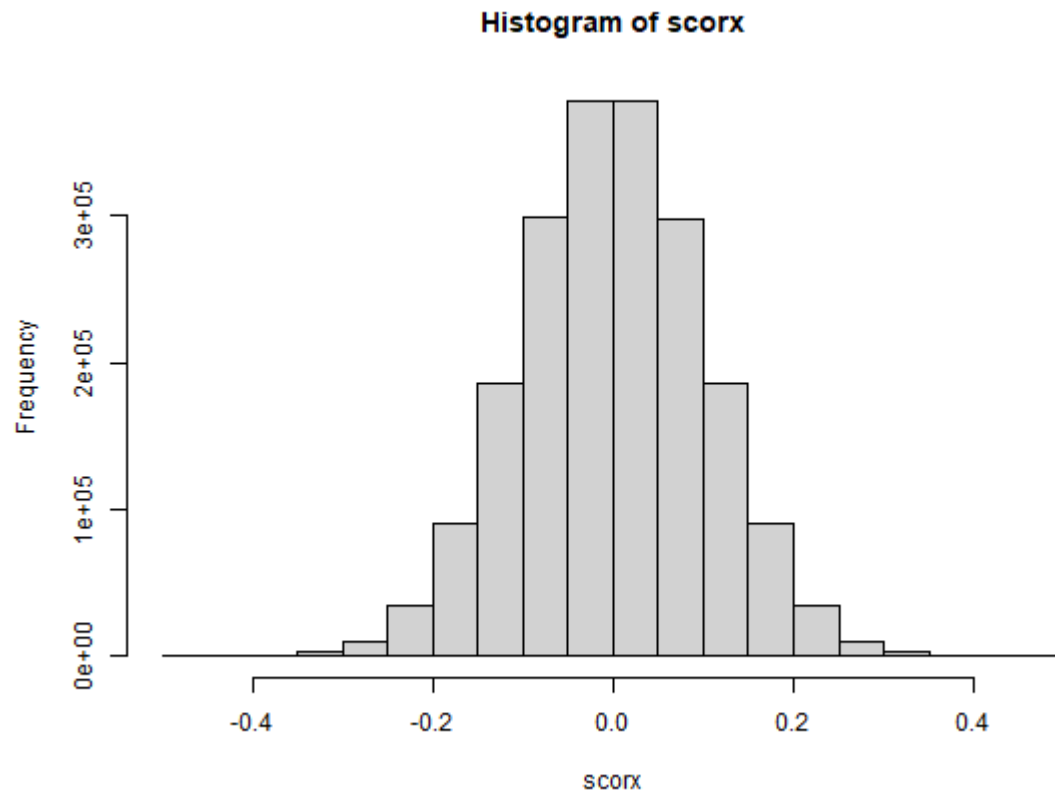
Let's generate some data with strong signals and low collinearity:

```
set.seed(1)  
n <- 100 # obs  
p <- 2000 # predictors  
pnot <- 5 # relevant predictors  
SNR <- 3 # => R^2 approx 0.8  
b <- rep(0, p)  
b[1:pnot] <- 0.5  
mu <- rep(0, p) # create a random matrix X (mean zero and  
# uncorrelated predictors with unit variance)  
sigma <- diag(p)  
X <- rmvnorm(n, mu, sigma)  
# strongest (spurious) correlations  
corx <- cor(X)  
corxtri <- corx[upper.tri(cor(X))]  
scorx <- sort(corxtri)
```

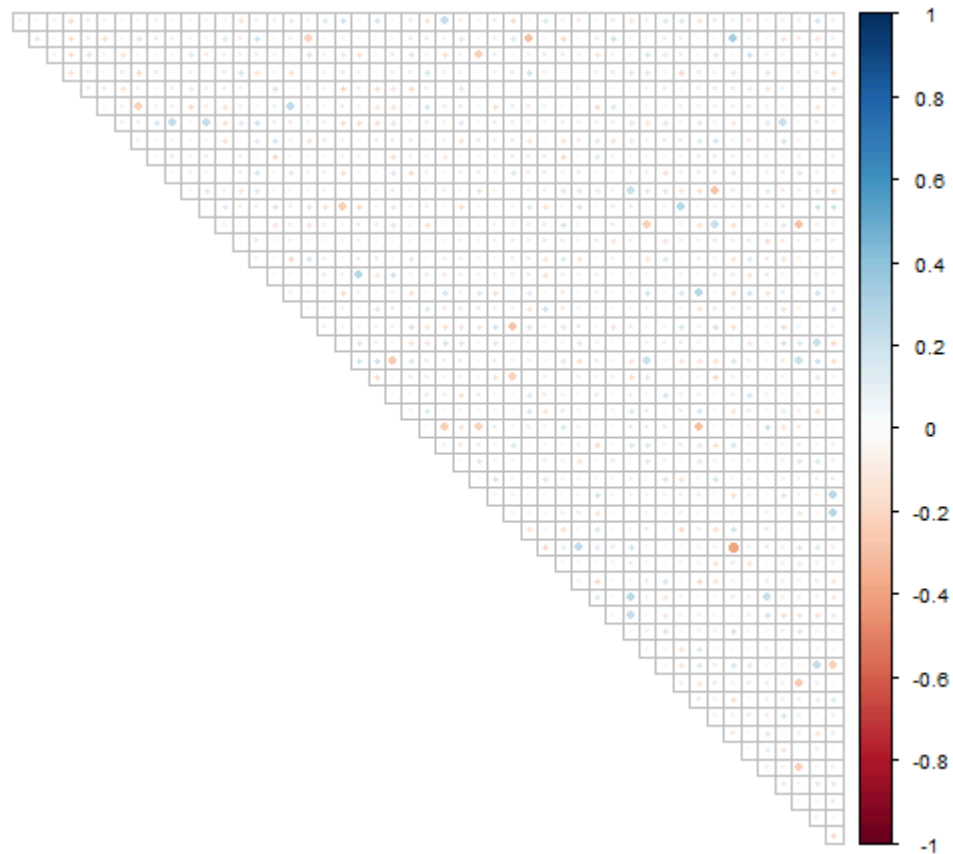
```
plot(1:(p*(p-1)/2), scorx, type="l")  
abline(h=0, col="red", lty=2)
```



```
hist(scorx)
```



```
# plot correlations in X (just take 50 predictors at random)
randp <- sample(1:p, 50)
corrplot(corx[randp,randp], tl.pos='n', type="upper", diag=F)
```



```
# true predictions
truepred <- X %*% b
# variance of the error term according to the SNR
varerr <- var(truepred)/SNR
# generate the error
err <- rnorm(n)*sqrt(varerr)
# create the response
y <- truepred + err
# sanity check
var(truepred)/var(err)
```

```
##           [,1]
## [1,] 3.754791
```

Let's check the **SIS** function. It first implements the Iterative Sure Independence Screening for different variants of (I)SIS, and then fits the final regression model using the R packages **ncvreg** and **glmnet** for the SCAD/MCP/LASSO regularized loglikelihood for the variables picked by (I)SIS. The main arguments are:

- **x**: The design matrix
- **y**: The response vector
- **penalty**: The penalty to be applied in the regularized likelihood subproblems. 'SCAD' (the default), 'MCP', or 'lasso' are provided.
- **varISIS**: Specifies whether to perform any of the two ISIS variants based on randomly splitting the sample into two groups. The variant `varISIS='aggr'` is an aggressive variable screening procedure, while `varISIS='cons'` is a more conservative approach. The default is `varISIS='vanilla'`, which performs the traditional vanilla version of ISIS.
- **iter**: Specifies whether to perform iterative SIS. The default is `iter=TRUE`.
- **iter.max**: Maximum number of iterations for (I)SIS and its variants.
- **tune**: Method for tuning the regularization parameter of the penalized likelihood subproblems and of the final model selected by (I)SIS. Options include `tune='bic'`, `tune='ebic'`, `tune='aic'`, and `tune='cv'`.
- **nfolds**: Number of folds used in cross-validation. The default is 10.
- **nsis**: Number of predictors recruited by (I)SIS.
- **standardize**: Logical flag for x variable standardization
- **seed**: Random seed used for sample splitting, random permutation, and cross-validation sampling of training and test sets.

We now perform variable selection using: - SIS: in its vanilla version - ISIS: the iterated version of SIS. These are paired with the LASSO penalty and the 10-fold cross-validation method for choosing the regularization parameter.

```
# num of features retained by SIS/ISIS
# q <- n-1
q <- round(n / log(n))
q
```

```
## [1] 22
```

```
# maximum num of iterations for ISIS
maxit <- 10
# vanilla SIS
model1 = SIS(X, y, penalty = "lasso",
varISIS = "vanilla", iter=F, tune = "cv", nfolds = 10,
nsis=q, standardize = TRUE, seed = 1)
```



```
options(width = 60)
# SIS selected features: 22
model1$sis.ix0
```

```
## [1]      1      2      3      4      5     24    240    288    332    343    717
## [12]   721   759   788   927   979  1302  1607  1635  1816  1868  1893
```

```
# SIS+LASSO selected features: 14
model1$ix
```

```
## [1]      1      2      3      4      5    240    288    721    788    979  1607
## [12]  1635  1868  1893
```

```
options(width = 60)
# vanilla ISIS
model2 = SIS(X, y,
  penalty = "lasso",
  varISIS = "vanilla", iter=T, iter.max=maxit,
  tune = "cv", nfolds = 10, nsis=q,
  standardize = FALSE, seed = 1)
```

```
## Iter 1 , screening:  1 2 3 4 5 240 717 721 788 979 1302 1607 1635 1893
## Iter 1 , selection:  1 2 3 4 5 240 721 788 979 1607 1635 1893
## Iter 1 , conditional-screening:  341 389 565 610 1073 1082 1113 1173 1338 1342
## Iter 2 , screening:  1 2 3 4 5 240 341 389 565 610 721 788 979 1073 1082 1113 117
## Iter 2 , selection:  1 2 3 4 5 240 341 389 565 610 721 788 979 1073 1082 1113 117
## Iter 2 , conditional-screening:  938
## Iter 3 , screening:  1 2 3 4 5 240 341 389 565 610 721 788 938 979 1073 1082 1113
## Iter 3 , selection:  1 2 3 4 5 240 341 389 565 610 721 788 938 979 1073 1082 1113
## Maximum number of variables selected
```

```
options(width = 60)
# SIS selected features: 14
model2$sis.ix0
```

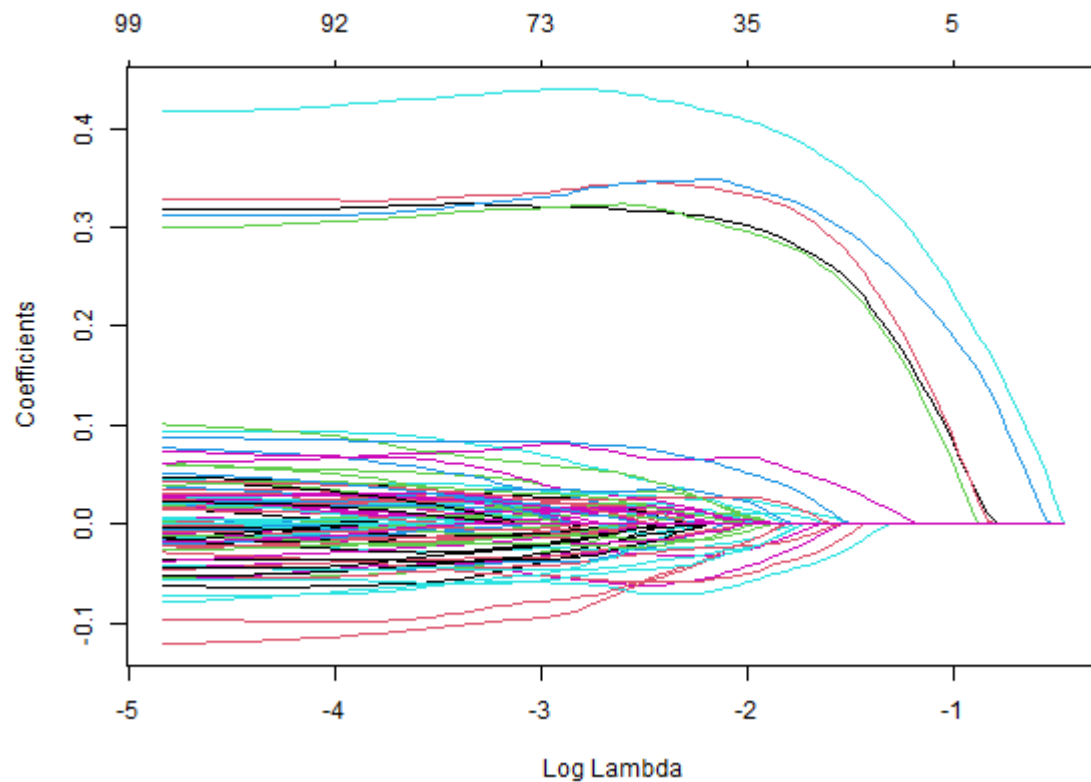
```
## [1]      1      2      3      4      5  240  717  721  788  979 1302
## [12] 1607 1635 1893
```

```
# ISIS+Lasso selected features: 22
model2$ix
```

```
## [1]      1      2      3      4      5  240  341  389  565  610  721
## [12]  788  938  979 1073 1082 1113 1173 1338 1342 1607 1635
```

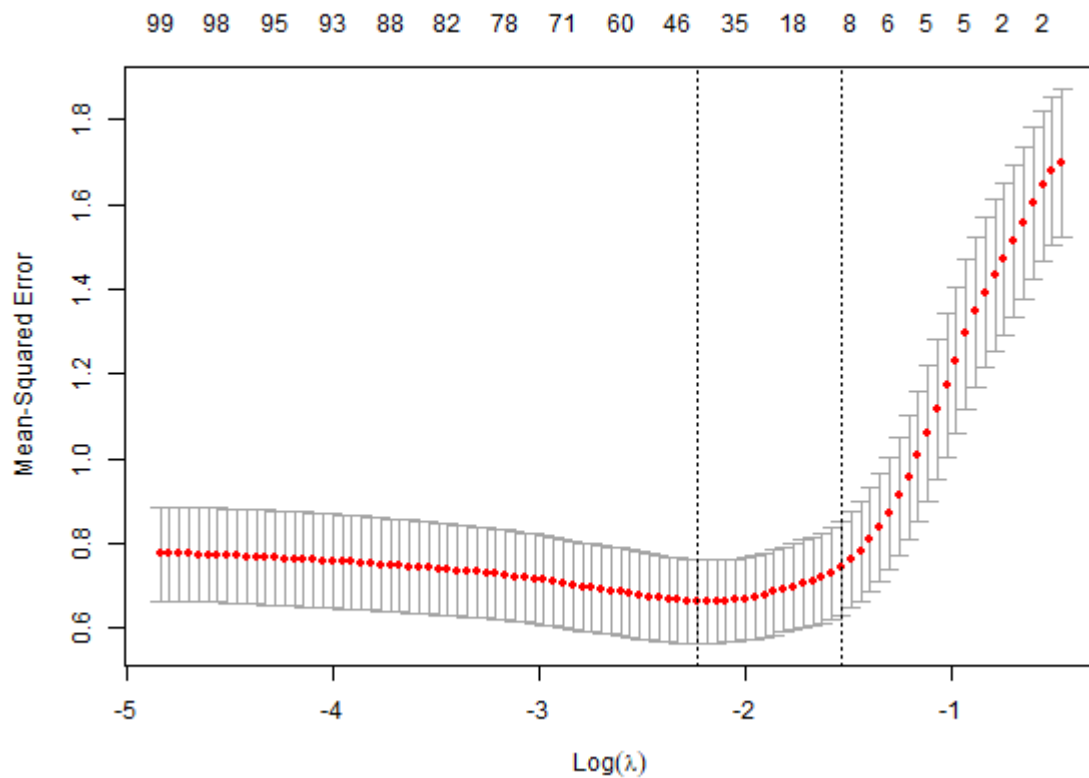
Compare it with a Lasso fit. Here is the overall path:

```
modelLasso = glmnet(X, y, standardize = T)  
plot(modelLasso, xvar="lambda")
```



Let's select the tuning parameter:

```
set.seed(1)  
modelLassoCV = cv.glmnet(X, y, standardize = T)  
plot(modelLassoCV)
```



Extract the associated features

```
modelLasso = glmnet(X, y, standardize = T,  
lambda=modelLassoCV$lambda.1se)  
Lassocoef <- which((coef(modelLasso)[-1]) != 0)  
Lassocoef
```

```
## [1] 1 2 3 4 5 288 788 979 1607 1728
```

```
# compare them with SIS  
Lassocoef %in% model1$ix
```

```
## [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE  
## [10] FALSE
```

```
# compare them with ISIS  
Lassocoef %in% model2$ix
```

```
## [1] TRUE TRUE TRUE TRUE TRUE FALSE TRUE TRUE TRUE  
## [10] FALSE
```

**Now it's your turn!!!**