Feature Screening

F. Chiaromonte (48.5%), L. Insolia (48.5%), L. Testa (1%)

March 23rd 2023

Contents

Introduction

Libraries

We are going to use:

```
# auto-install SIS if needed
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
library(tidyverse)  # data manipulation and visualization
library(caret)      # classification learning (confusion matrix)
```

Data

We will simulate some Gaussian data for an example with continuous response.

We will also analyze Leukemia data (with a binary response) that are provided in the SIS package: - leukemia.test: Gene expression 7129 genes from 34 patients with acute leukemias (20 in class Acute Lymphoblastic Leukemia and 14 in class Acute Myeloid Leukemia). - leukemia.train: Gene expression of 7129 genes from 38 patients with acute leukemias (27 in class Acute Lymphoblastic Leukemia and 11 in class Acute Myeloid Leukemia).

Linear regression

"Ideal" scenario

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

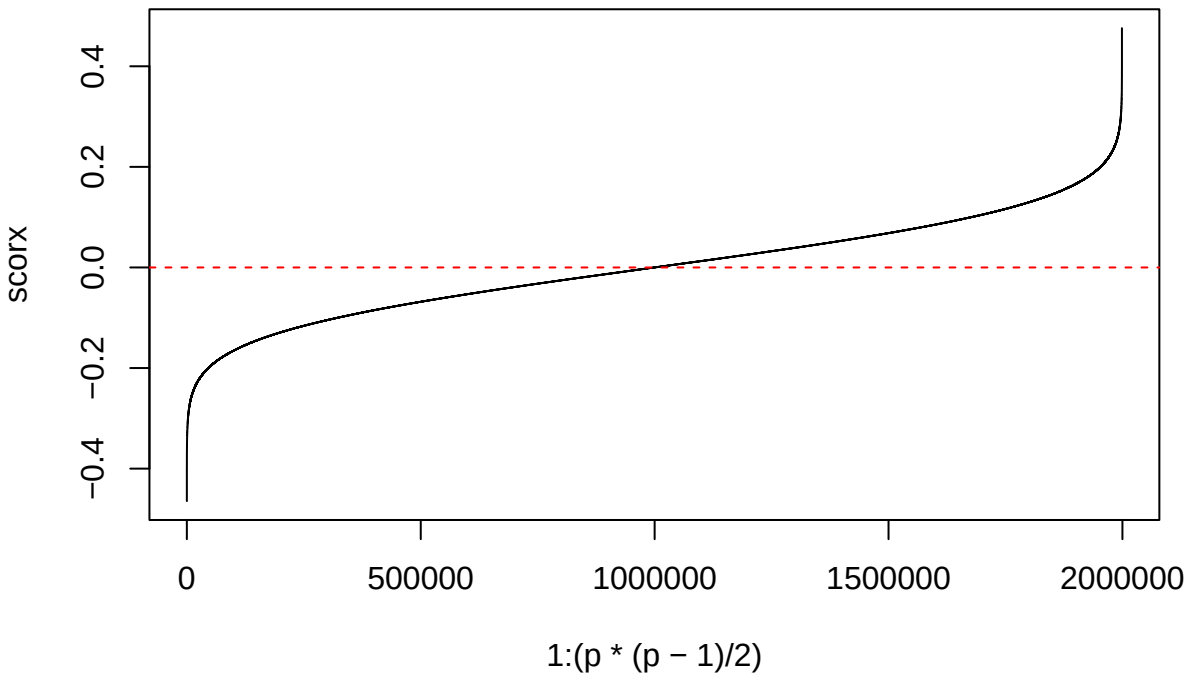
```r
set.seed(1) # Do you remember this command???

n <- 100   # obs
p <- 2000  # predictors
pnot <- 5  # relevant predictors

# signal to noise ratio SNR = var(Xb)/var(err) -- similar to R^2 but unbounded
SNR <- 3  # => R^2 approx 0.8

# regression coefficients
b <- rep(0, p)
b[1:pnot] <- 0.5

# create a random matrix X (mean zero and uncorrelated predictors with unit variance)
mu <- rep(0, p)
sigma <- diag(p)
X <- rmvnorm(n, mu, sigma)

# strongest (spurious) correlations
corx <- cor(X)
corxtri <- corx[upper.tri(cor(X))]
scorx <- sort(corxtri)
head(scorx)
```

```
## [1] -0.4644961 -0.4520914 -0.4506670 -0.4423717 -0.4404513 -0.4375935
```

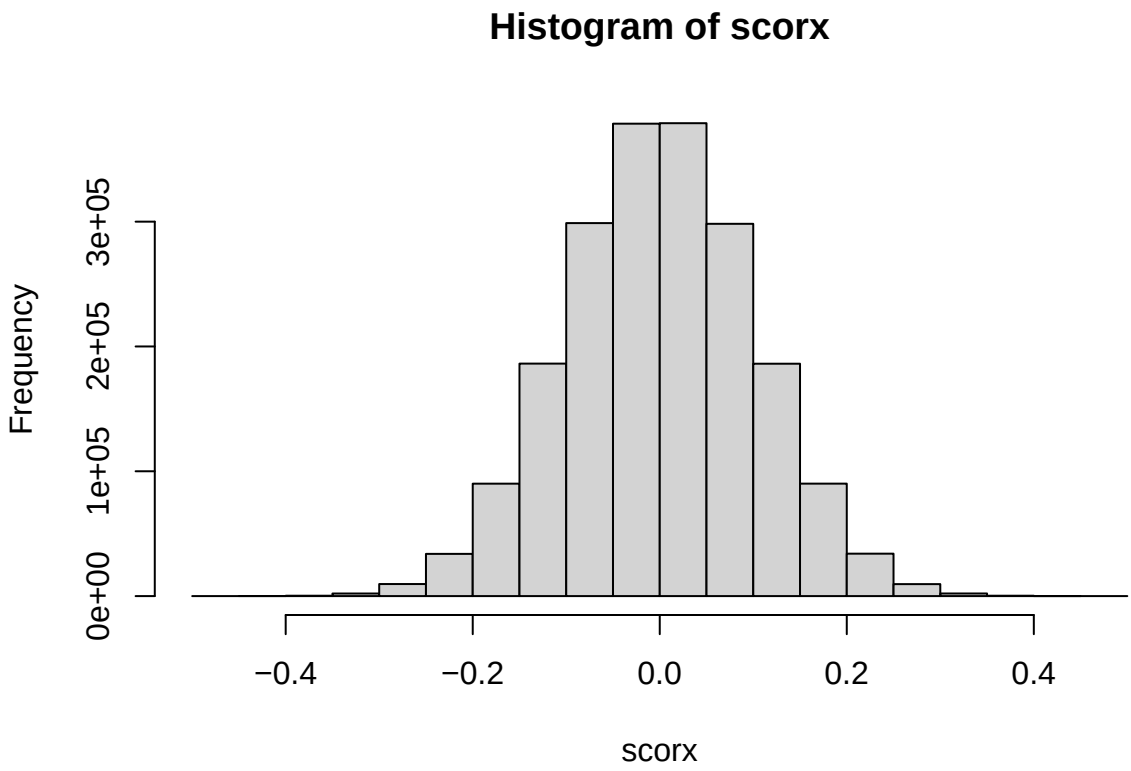
```r
tail(scorx)
```

```
## [1] 0.4293648 0.4406117 0.4484676 0.4519254 0.4671771 0.4757770
```

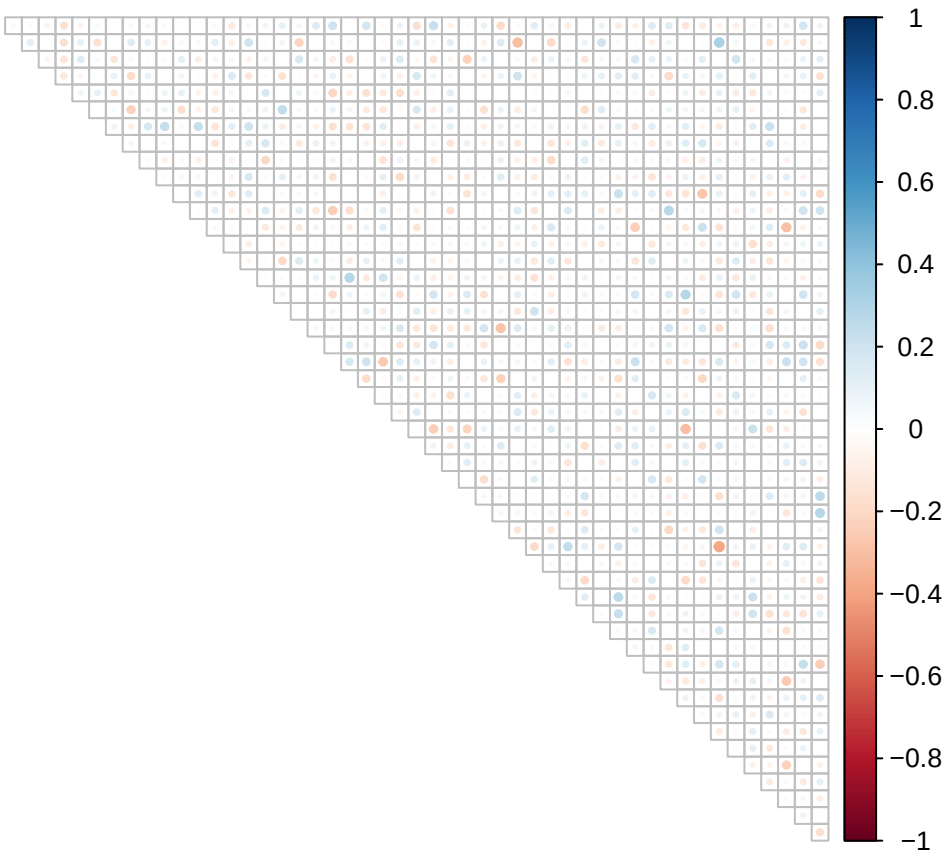
```r
plot(1:(p*(p-1)/2), scorx, type="l")
abline(h=0, col="red", lty=2)
```



```
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
varerr
```

```
##           [,1]
## [1,] 0.4041815
```

```
var(err)
```

```
## [1] 0.3229326
```

```
summary(lm(y ~ X[, 1:pnot]))
```

```
##
## Call:
## lm(formula = y ~ X[, 1:pnot])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.66704 -0.35200 -0.04304  0.33877  1.48280
```

```
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.10240    0.05835   1.755   0.0825 .
## X[, 1:pnot]1  0.49884    0.05775   8.639 1.44e-13 ***
## X[, 1:pnot]2  0.57419    0.05935   9.674 9.05e-16 ***
## X[, 1:pnot]3  0.54383    0.05833   9.324 5.03e-15 ***
## X[, 1:pnot]4  0.45946    0.06120   7.508 3.40e-11 ***
## X[, 1:pnot]5  0.57151    0.05562  10.275  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5717 on 94 degrees of freedom
## Multiple R-squared:  0.813,  Adjusted R-squared:  0.803
## F-statistic: 81.71 on 5 and 94 DF,  p-value: < 2.2e-16
```

```
# summary(lm(y ~ X[, 1:p]))

y.train <- y
x.train <- X
```

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.

```
help(SIS)
```

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 ten-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.train, y.train,
             penalty = "lasso",
             varISIS = "vanilla", iter=F,
             tune = "cv", nfolds = 10, nsis=q,
             standardize = FALSE, seed = 1)

# vanilla ISIS
model2 = SIS(x.train, y.train,
             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 1173 1338 1342 1607 163
## Iter 2 , selection:  1 2 3 4 5 240 341 389 565 610 721 788 979 1073 1082 1113 1173 1338 1342 1607 163
## 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 1173 1338 1342 160
## Iter 3 , selection:  1 2 3 4 5 240 341 389 565 610 721 788 938 979 1073 1082 1113 1173 1338 1342 160
## Maximum number of variables selected
```

Lets'see the selected predictors:

```
cat("\n\n\n")
```

```
print(paste0("SIS selected features: ", length(model1$ix0)))
```

```
## [1] "SIS selected features: 22"
```

```
model1$ix0
```

```
##  [1]    1    2    3    4    5   24  240  288  332  343  717  721  759  788  927
## [16]  979 1302 1607 1635 1816 1868 1893
```

```
cat("\n\n\n")
```

```
print(paste0("SIS+Lasso selected features: ", length(model1$ix)))
```

```
## [1] "SIS+Lasso selected features: 14"
```

```
model1$ix
```

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

```
cat("\n\n\n")
```

```
print(paste0("ISIS selected features: ", length(model2$ix0)))
```

```
## [1] "ISIS selected features: 22"
```

```
model2$ix0
```

```
##  [1]    1    2    3    4    5  240  341  389  565  610  721  788  938  979 1073
## [16] 1082 1113 1173 1338 1342 1607 1635
```

```
cat("\n\n\n")
```

```
print(paste0("ISIS+Lasso selected features: ", length(model2$ix)))
```

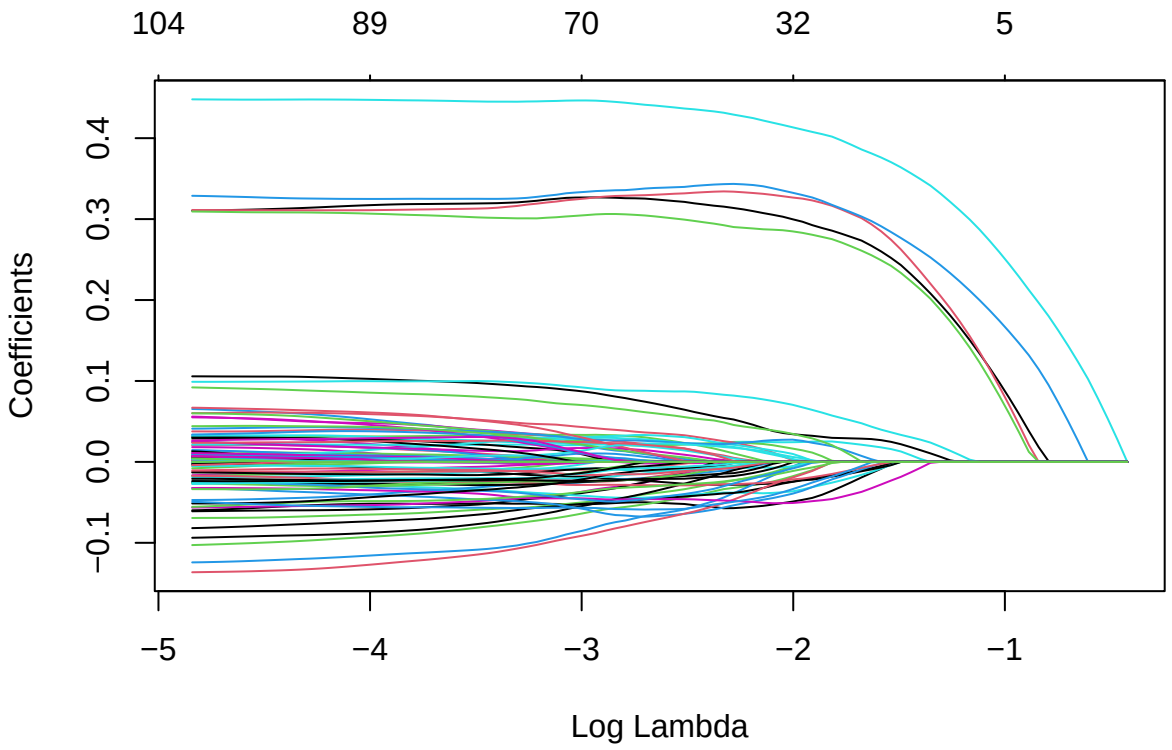
```
## [1] "ISIS+Lasso selected features: 22"
```

```
model2$ix
```

```
##  [1]    1    2    3    4    5  240  341  389  565  610  721  788  938  979 1073
## [16] 1082 1113 1173 1338 1342 1607 1635
```

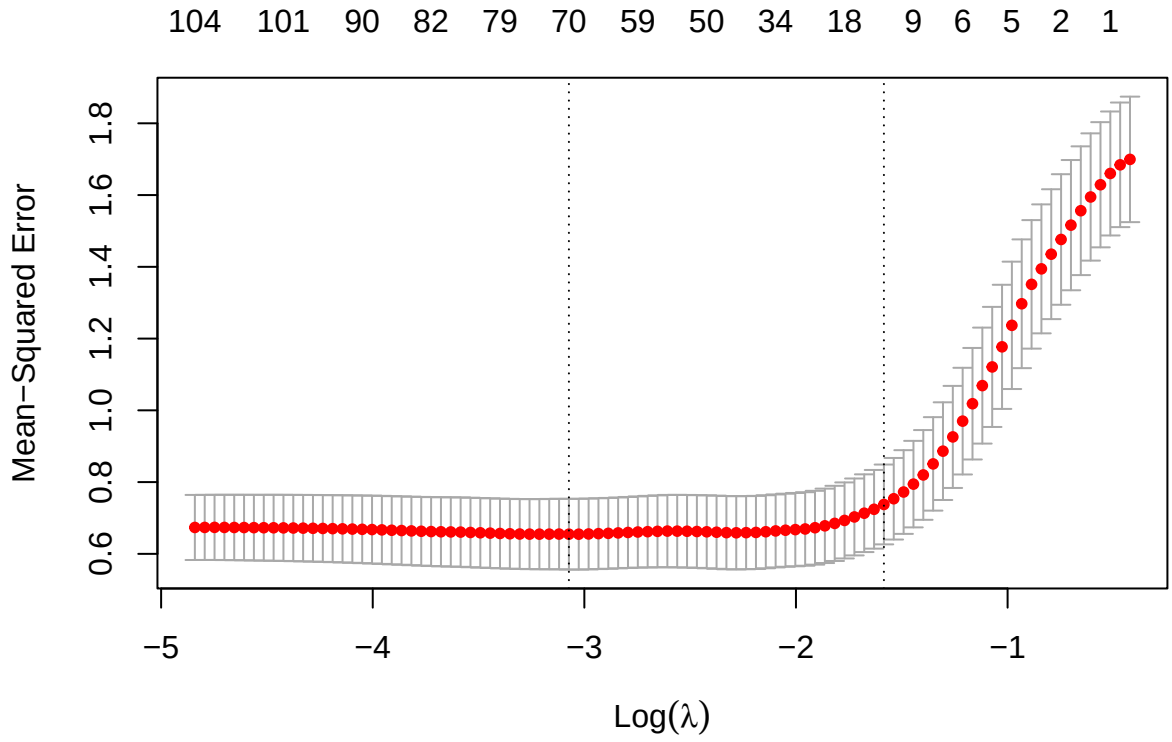
Compare it with a Lasso fit. Here is the overall path:

```
modelLasso = glmnet(x.train, y.train, standardize = F)
plot(modelLasso, xvar="lambda")
```



Let's select the tuning parameter:

```
set.seed(1)

modelLassoCV = cv.glmnet(x.train, y.train, standardize = F)
plot(modelLassoCV)
```



Extract the associated features

```
modelLasso = glmnet(x.train, y.train, standardize = F,
                    lambda=modelLassoCV$lambda.1se)
Lassocoef <- which(coef(modelLasso) != 0)
Lassocoef
```

```
##  [1]    1    2    3    4    5    6   30  144  289  454  613  722  789  980 1010
## [16] 1608
```

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

```
##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [13] FALSE FALSE FALSE FALSE
```

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

```
##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [13] FALSE FALSE FALSE FALSE
```

Let's see if we can retrieve the "truly" relevant features using nonconvex penalization methods (MCP):

```
# vanilla SIS
model1 = SIS(x.train, y.train,
             penalty = "MCP",
             varISIS = "vanilla", iter=F,
             tune = "cv", nsis=q,
             standardize = FALSE, seed = 1)

# vanilla ISIS
model2 = SIS(x.train, y.train,
             penalty = "MCP",
             varISIS = "vanilla", iter=T, iter.max=maxit,
             tune = "cv", 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
## Iter 1 , conditional-screening:  170 300 389 412 767 911 924 1005 1073 1112 1113 1173 1221 1338 1398
## Iter 2 , screening:  1 2 3 4 5 170 300 389 412 767 911 924 1005 1073 1112 1113 1173 1221 1338 1398 15
## Iter 2 , selection:  1 2 3 4 5 170 300 389 412 767 924 1073 1112 1113 1173 1338 1398 1551 1753
## Iter 2 , conditional-screening:  367 955 1286
## Iter 3 , screening:  1 2 3 4 5 170 300 367 389 412 767 924 955 1073 1112 1113 1173 1286 1338 1398 155
## Iter 3 , selection:  1 2 3 4 5 170 300 367 389 412 767 924 955 1112 1113 1173 1286 1338 1551 1753
## Iter 3 , conditional-screening:  715 1057
## Iter 4 , screening:  1 2 3 4 5 170 300 367 389 412 715 767 924 955 1057 1112 1113 1173 1286 1338 155
## Iter 4 , selection:  1 2 3 4 5 170 300 367 389 412 715 924 955 1057 1112 1113 1173 1286 1338 1551 175
## Iter 4 , conditional-screening:  1295
## Iter 5 , screening:  1 2 3 4 5 170 300 367 389 412 715 924 955 1057 1112 1113 1173 1286 1295 1338 155
## Iter 5 , selection:  1 2 3 4 5 170 300 367 412 715 924 955 1057 1112 1113 1173 1286 1295 1338 1551 17
## Iter 5 , conditional-screening:  656
## Iter 6 , screening:  1 2 3 4 5 170 300 367 412 656 715 924 955 1057 1112 1113 1173 1286 1295 1338 155
## Iter 6 , selection:  1 2 3 4 5 170 300 367 412 656 715 924 955 1057 1112 1113 1173 1286 1295 1338 155
## Maximum number of variables selected
```

Let's see the results:

```
cat("\n\n\n")
```

```
print(paste0("SIS selected features: ", length(model1$ix0)))
```

```
## [1] "SIS selected features: 22"
```

```
model1$ix0
```

```
##  [1]    1    2    3    4    5   24  240  288  332  343  717  721  759  788  927
## [16]  979 1302 1607 1635 1816 1868 1893
```

```
cat("\n\n\n")
```

```
print(paste0("SIS+MCP selected features: ", length(model1$ix)))
```

```
## [1] "SIS+MCP selected features: 5"
```

```
model1$ix
```

```
## [1] 1 2 3 4 5
```

```
cat("\n\n\n")
```

```
print(paste0("ISIS selected features: ", length(model2$ix0)))
```

```
## [1] "ISIS selected features: 22"
```

```
model2$ix0
```

```
##  [1]    1    2    3    4    5  170  300  367  412  656  715  924  955 1057 1112
## [16] 1113 1173 1286 1295 1338 1551 1753
```

```
cat("\n\n\n")
```

```
print(paste0("ISIS+MCP selected features: ", length(model2$ix)))
```

```
## [1] "ISIS+MCP selected features: 22"
```

```
model2$ix
```

```
##  [1]    1    2    3    4    5  170  300  367  412  656  715  924  955 1057 1112
## [16] 1113 1173 1286 1295 1338 1551 1753
```

Stronger collinearity

Let's generate some data as before, but with stronger collinearity. Namely, Σ_x has an autoregressive correlation structure, where $\Sigma_{x,ij} = \rho^{|i-j|}$ for $\rho = 0.6$:

```
set.seed(1)

n <- 100   # obs
p <- 2000  # predictors
pnot <- 5  # relevant predictors

# signal to noise ratio SNR = var(Xb)/var(err) -- similar to R^2 but unbounded
SNR <- 3  # => R^2 approx 0.8

# regression coefficients
b <- rep(0, p)
b[1:pnot] <- 0.5
```

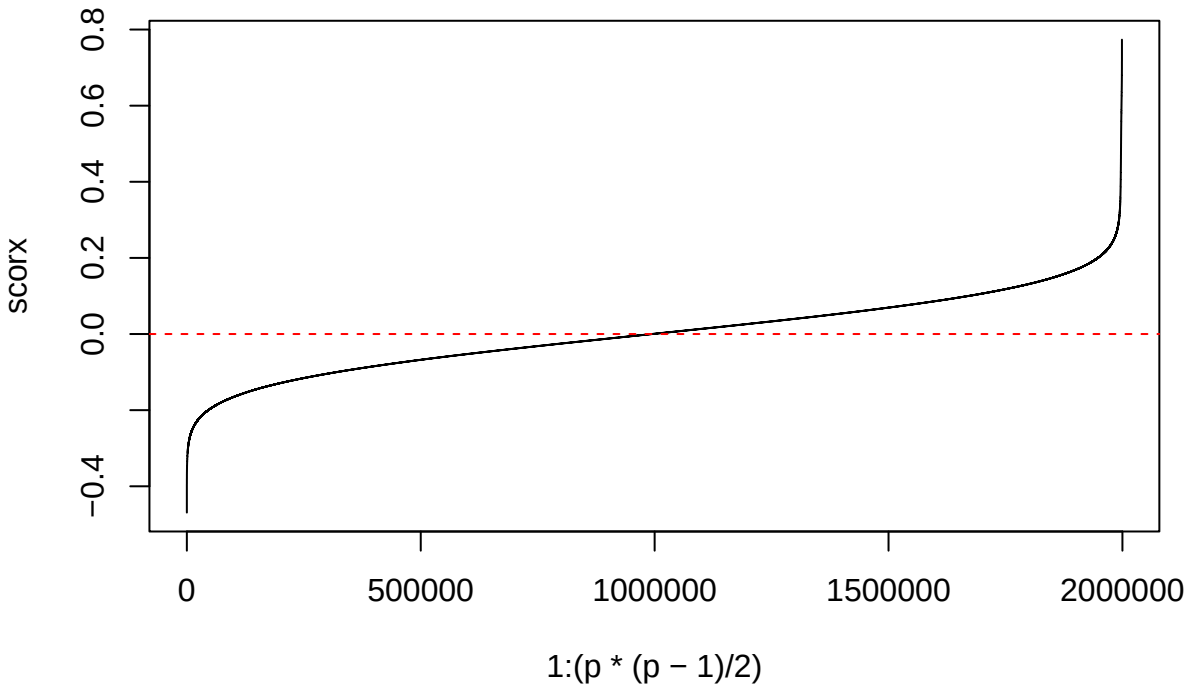
```r
# create a random matrix X (mean zero and autoregressive correlation)
rho <- 0.6
mu <- rep(0, p)
sigma <- matrix(NA, p, p)
for (i in 1:p) {
  for (j in 1:p) {
    sigma[i,j] <- rho^abs(i-j)
  }
}
X <- rmvnorm(n, mu, sigma)

# strongest correlations
corx <- cor(X)
corxtri <- corx[upper.tri(cor(X))]
scorx <- sort(corxtri)
head(scorx)
```

```
## [1] -0.4690702 -0.4584491 -0.4486408 -0.4442743 -0.4437989 -0.4400888
```

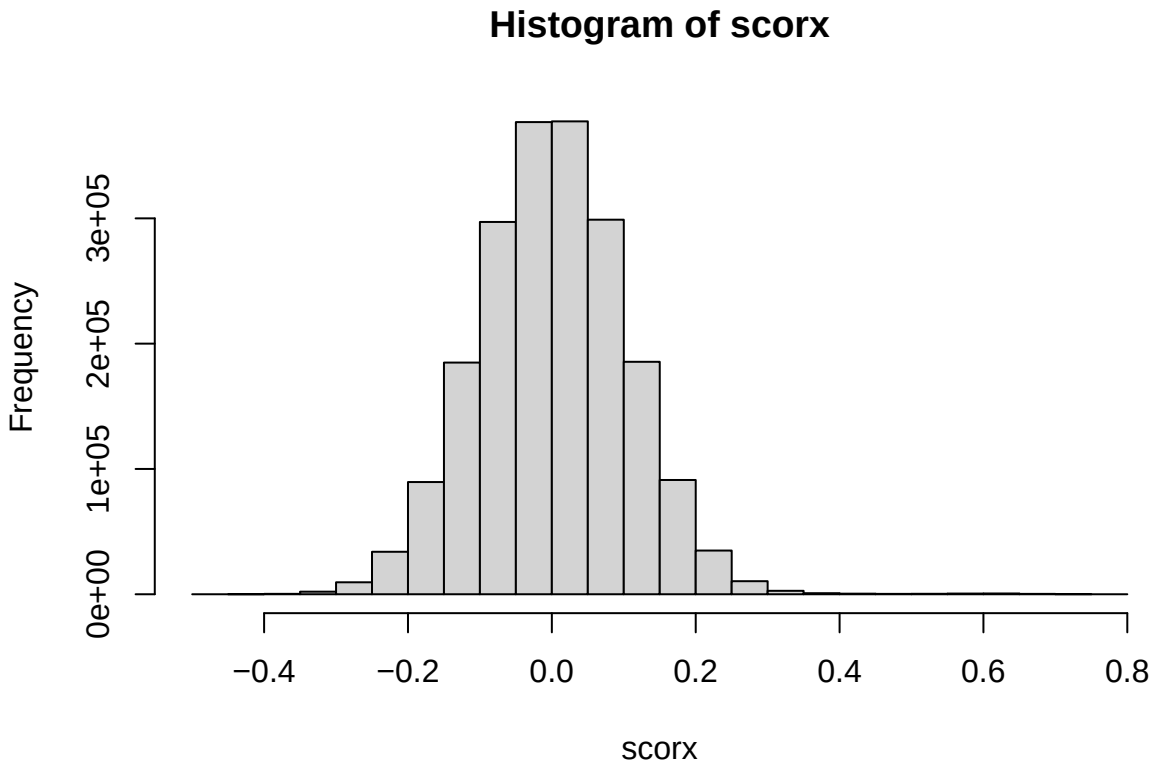
```r
tail(scorx)
```

```
## [1] 0.7539591 0.7546250 0.7558628 0.7625582 0.7640708 0.7733425
```

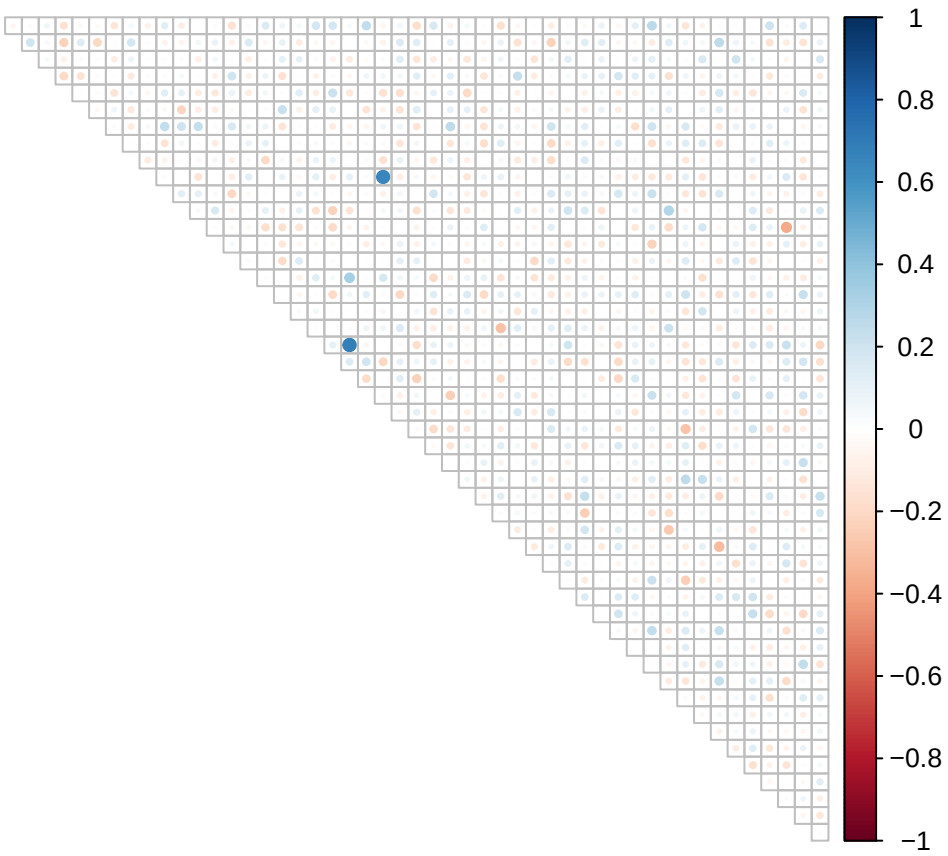
```r
plot(1:(p*(p-1)/2), scorx, type="l")
abline(h=0, col="red", lty=2)
```




```r
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
varerr
```

```
##          [,1]
## [1,] 1.030342
```

```
var(err)
```

```
## [1] 0.8232217
```

```
summary(lm(y ~ X[, 1:pnot]))
```

```
##
## Call:
## lm(formula = y ~ X[, 1:pnot])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.59205 -0.57396 -0.03759  0.56478  2.35761
```

```
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.17199    0.09366   1.836 0.069463 .
## X[, 1:pnot]1  0.45385    0.11176   4.061 0.000101 ***
## X[, 1:pnot]2  0.64264    0.13024   4.934 3.47e-06 ***
## X[, 1:pnot]3  0.56358    0.13564   4.155 7.16e-05 ***
## X[, 1:pnot]4  0.35076    0.15108   2.322 0.022409 *
## X[, 1:pnot]5  0.59838    0.11944   5.010 2.55e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9182 on 94 degrees of freedom
## Multiple R-squared:  0.8054, Adjusted R-squared:  0.795
## F-statistic: 77.79 on 5 and 94 DF,  p-value: < 2.2e-16
```

```
# summary(lm(y ~ X[, 1:p]))

y.train <- y
x.train <- X
```

We will now replicate the same analysis as before:

```
# 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.train, y.train,
             penalty = "lasso",
             varISIS = "vanilla", iter=F,
             tune = "cv", nfolds = 10, nsis=q,
             standardize = FALSE, seed = 1)

# vanilla ISIS
model2 = SIS(x.train, y.train,
             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 6 518 519 979 1606 1607 1608 1816 1893
## Iter 1 , selection:  1 2 3 4 5 979 1607 1608 1893
## Iter 1 , conditional-screening:  565 922 923 924 1112 1113 1173 1295 1338 1602 1867 1868 1997
## Iter 2 , screening:  1 2 3 4 5 565 922 923 924 979 1112 1113 1173 1295 1338 1602 1607 1608 1867 1868
## Iter 2 , selection:  1 2 3 4 5 565 922 924 979 1112 1113 1173 1295 1338 1602 1607 1867 1868 1893 199
## Iter 2 , conditional-screening:  579 938
## Iter 3 , screening:  1 2 3 4 5 565 579 922 924 938 979 1112 1113 1173 1295 1338 1602 1607 1867 1868
## Iter 3 , selection:  1 2 3 4 5 565 579 922 924 938 979 1112 1113 1173 1295 1338 1602 1607 1867 1893
## Iter 3 , conditional-screening:  564
```

```
## Iter 4 , screening:  1 2 3 4 5 564 565 579 922 924 938 979 1112 1113 1173 1295 1338 1602 1607 1867 18
## Iter 4 , selection:  1 2 3 4 5 564 565 579 922 924 938 979 1112 1113 1173 1295 1338 1602 1607 1867 18
## Maximum number of variables selected
```

Lets'see the selected predictors:

```
cat("\n\n\n")
```

```
print(paste0("SIS selected features: ", length(model1$ix0)))
```

```
## [1] "SIS selected features: 22"
```

```
model1$ix0
```

```
##  [1]    1    2    3    4    5    6  240  318  319  518  519  927  979 1009 1246
## [16] 1605 1606 1607 1608 1635 1816 1893
```

```
cat("\n\n\n")
```

```
print(paste0("SIS+Lasso selected features: ", length(model1$ix)))
```

```
## [1] "SIS+Lasso selected features: 9"
```

```
model1$ix
```

```
## [1]    1    2    3    4    5  319  979 1607 1608
```

```
cat("\n\n\n")
```

```
print(paste0("ISIS selected features: ", length(model2$ix0)))
```

```
## [1] "ISIS selected features: 22"
```

```
model2$ix0
```

```
##  [1]    1    2    3    4    5  564  565  579  922  924  938  979 1112 1113 1173
## [16] 1295 1338 1602 1607 1867 1893 1997
```

```
cat("\n\n\n")
```

```
print(paste0("ISIS+Lasso selected features: ", length(model2$ix)))
```

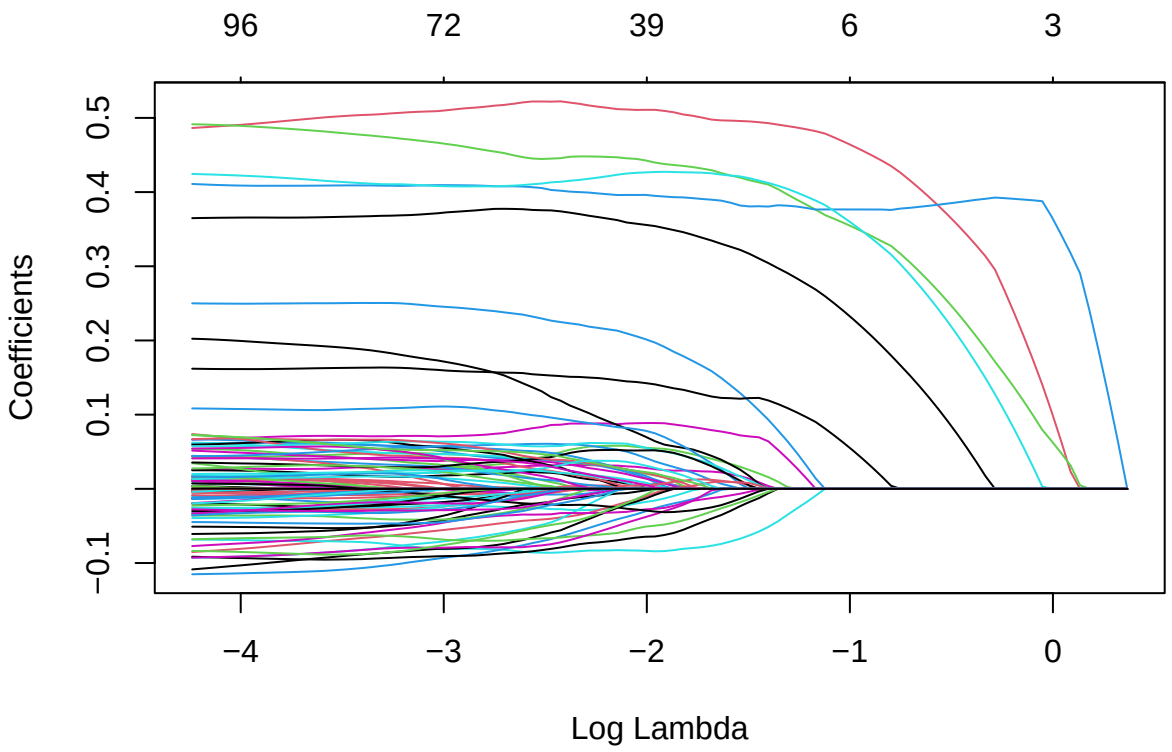
```
## [1] "ISIS+Lasso selected features: 22"
```

```
model2$ix
```

```
##  [1]    1    2    3    4    5  564  565  579  922  924  938  979 1112 1113 1173
## [16] 1295 1338 1602 1607 1867 1893 1997
```

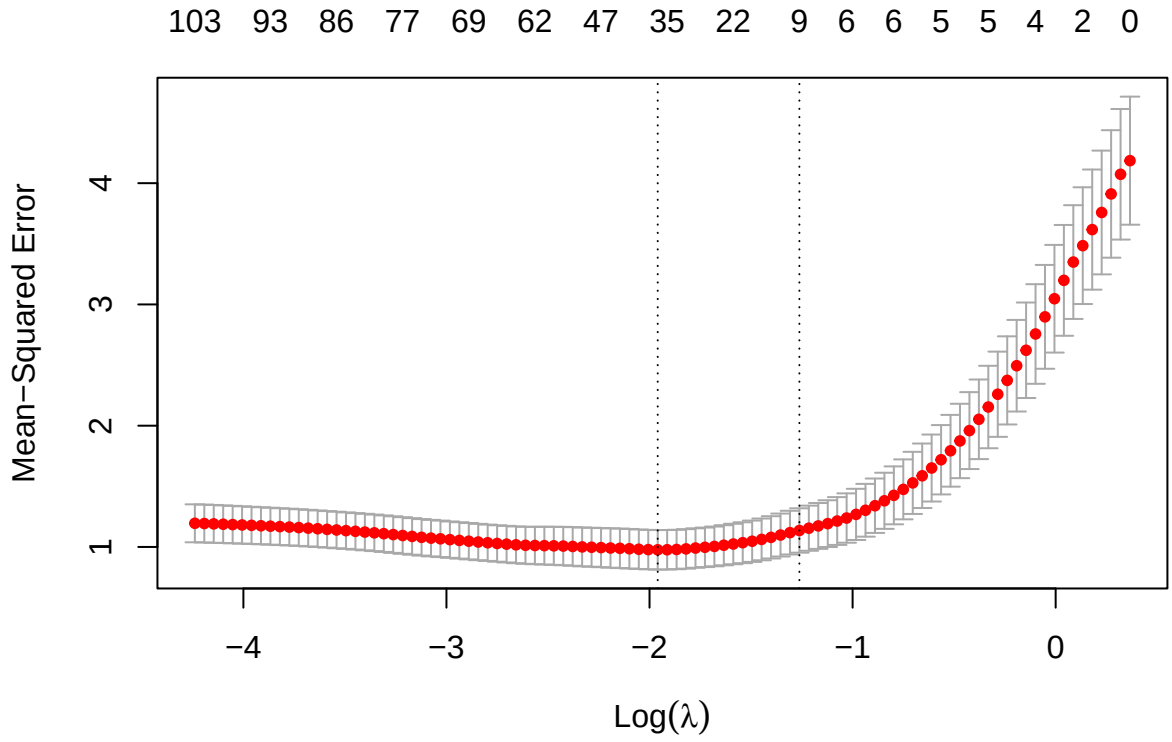
Compare it with a Lasso fit. Here is the overall path:

```
modelLasso = glmnet(x.train, y.train, standardize = F)
plot(modelLasso, xvar="lambda")
```



Let's select the tuning parameter:

```
set.seed(1)

modelLassoCV = cv.glmnet(x.train, y.train, standardize = F)
plot(modelLassoCV)
```



Extract the associated features

```
modelLasso = glmnet(x.train, y.train, standardize = F,
                    lambda=modelLassoCV$lambda.1se)
Lassocoef <- which(coef(modelLasso) != 0)
Lassocoef
```

```
##  [1]    1    2    3    4    5    6  722  925 1113 1608
```

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

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

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

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

Let's see if we can retrieve the "truly" relevant features using nonconvex penalization methods (MCP):

```
# vanilla SIS
model1 = SIS(x.train, y.train,
             penalty = "MCP",
             varISIS = "vanilla", iter=F,
             tune = "cv", nsis=q,
             standardize = FALSE, seed = 1)

# vanilla ISIS
model2 = SIS(x.train, y.train,
             penalty = "MCP",
             varISIS = "vanilla", iter=T, iter.max=maxit,
             tune = "cv", nsis=q,
             standardize = FALSE, seed = 1)
```

```
## Iter 1 , screening:  1 2 3 4 5 6 518 519 979 1606 1607 1608 1816 1893
## Iter 1 , selection:  1 2 3 4 5 6 979 1608
## Iter 1 , conditional-screening:  922 923 924 1112 1113 1173 1295 1338 1602 1753 1782 1867 1868 1997
## Iter 2 , screening:  1 2 3 4 5 6 922 923 924 979 1112 1113 1173 1295 1338 1602 1608 1753 1782 1867 18
## Iter 2 , selection:  1 2 3 4 5 6 922 924 979 1112 1113 1173 1295 1338 1608 1753 1782 1867 1997
## Iter 2 , conditional-screening:  579 938 1607
## Iter 3 , screening:  1 2 3 4 5 6 579 922 924 938 979 1112 1113 1173 1295 1338 1607 1608 1753 1782 18
## Iter 3 , selection:  1 2 3 4 5 579 924 938 979 1112 1113 1173 1295 1338 1607 1782 1867 1997
## Iter 3 , conditional-screening:  643 703 1286 1418
## Iter 4 , screening:  1 2 3 4 5 579 643 703 924 938 979 1112 1113 1173 1286 1295 1338 1418 1607 1782
## Iter 4 , selection:  1 2 3 4 5 579 643 703 924 938 979 1112 1113 1173 1286 1295 1338 1418 1607 1782
## Maximum number of variables selected
```

Let's see the results:

```
cat("\n\n\n")
```

```
print(paste0("SIS selected features: ", length(model1$ix0)))
```

```
## [1] "SIS selected features: 22"
```

```
model1$ix0
```

```
##  [1]    1    2    3    4    5    6  240  318  319  518  519  927  979 1009 1246
## [16] 1605 1606 1607 1608 1635 1816 1893
```

```
cat("\n\n\n")
```

```
print(paste0("SIS+MCP selected features: ", length(model1$ix)))
```

```
## [1] "SIS+MCP selected features: 8"
```

```
model1$ix
```

```
## [1]    1    2    3    4    5    6  319 1608
```

```
cat("\n\n\n")
```

```
print(paste0("ISIS selected features: ", length(model2$ix0)))
```

```
## [1] "ISIS selected features: 22"
```

```
model2$ix0
```

```
##  [1]    1    2    3    4    5  579  643  703  924  938  979 1112 1113 1173 1286
## [16] 1295 1338 1418 1607 1782 1867 1997
```

```
cat("\n\n\n")
```

```
print(paste0("ISIS+MCP selected features: ", length(model2$ix)))
```

```
## [1] "ISIS+MCP selected features: 22"
```

```
model2$ix
```

```
##  [1]    1    2    3    4    5  579  643  703  924  938  979 1112 1113 1173 1286
## [16] 1295 1338 1418 1607 1782 1867 1997
```

Logistic regression

```
set.seed(12345)
data("leukemia.train", package = "SIS")
data("leukemia.test", package = "SIS")
```

Let's construct our response variable and the design matrix.

```
y1 <- leukemia.train[, dim(leukemia.train)[2]]
x1 <- as.matrix(leukemia.train[, -dim(leukemia.train)[2]])

x2 <- as.matrix(leukemia.test[, -dim(leukemia.test)[2]])
y2 <- leukemia.test[, dim(leukemia.test)[2]]
```

We further combine the training and test samples and then perform a 50%–50% random splitting of the observed data into new training and test data for which the number of cases remains balanced across these new samples (i.e. balanced sample splitting).

In this manner, the balanced training and test samples are of size 36.

```
x <- rbind(x1, x2)
y <- c(y1, y2)
n <- dim(x)[1]
aux <- 1:n
ind.train1 <- sample(aux[y == 0], 23, replace = FALSE)
ind.train2 <- sample(aux[y == 1], 13, replace = FALSE)
ind.train <- c(ind.train1, ind.train2)
```

```
ind.test1 <- setdiff(aux[y == 0], ind.train1)
ind.test2 <- setdiff(aux[y == 1], ind.train2)
ind.test <- c(ind.test1, ind.test2)
```

Before variable screening and classification, we also standardize each predictor to zero mean and unit variance:

```
y.train <- y[ind.train]
y.test <- y[ind.test]

x.train <- scale(x[ind.train, ])
x.test <- scale(x[ind.test, ])
```

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 ten-fold cross-validation method for choosing the regularization parameter.

```
# vanilla SIS
model1 = SIS(x.train, y.train,
             family = "binomial",
             type.measure='auc',
             penalty = "lasso",
             varISIS = "vanilla", iter=F,
             tune = "cv", nfolds = 10, nsis=100,
             standardize = FALSE, seed = 9)

# vanilla ISIS
model2 = SIS(x.train, y.train,
             family = "binomial",
             type.measure='auc',
             penalty = "lasso",
             varISIS = "vanilla", iter=T, iter.max=2,
             tune = "cv", nfolds = 10, nsis=100,
             standardize = FALSE, seed = 9)
```

```
## Iter 1 , screening:  134 161 173 283 1144 1194 1615 1674 1745 1779 1829 1834 1882 2020 2121 2242 2248
## Iter 1 , selection:  134 1144 1834 2242 3433 4381 4847 4951 5335 6221
## Iter 1 , conditional-screening:  161 163 218 235 259 292 563 614 649 662 671 706 760 778 874 922 1093
## Iter 2 , screening:  134 161 163 218 235 259 292 563 614 649 662 671 706 760 778 874 922 1093 1108 1
## Iter 2 , selection:  134 1144 1779 1834 2242 3433 4381 4847 4951 5335 6221
## Maximum number of iterations reached
```

Lets'see the selected predictors:

```
model1$ix
```

```
##  [1]  134 1144 1834 2242 3433 4847 4951 5335 5766 6221
```

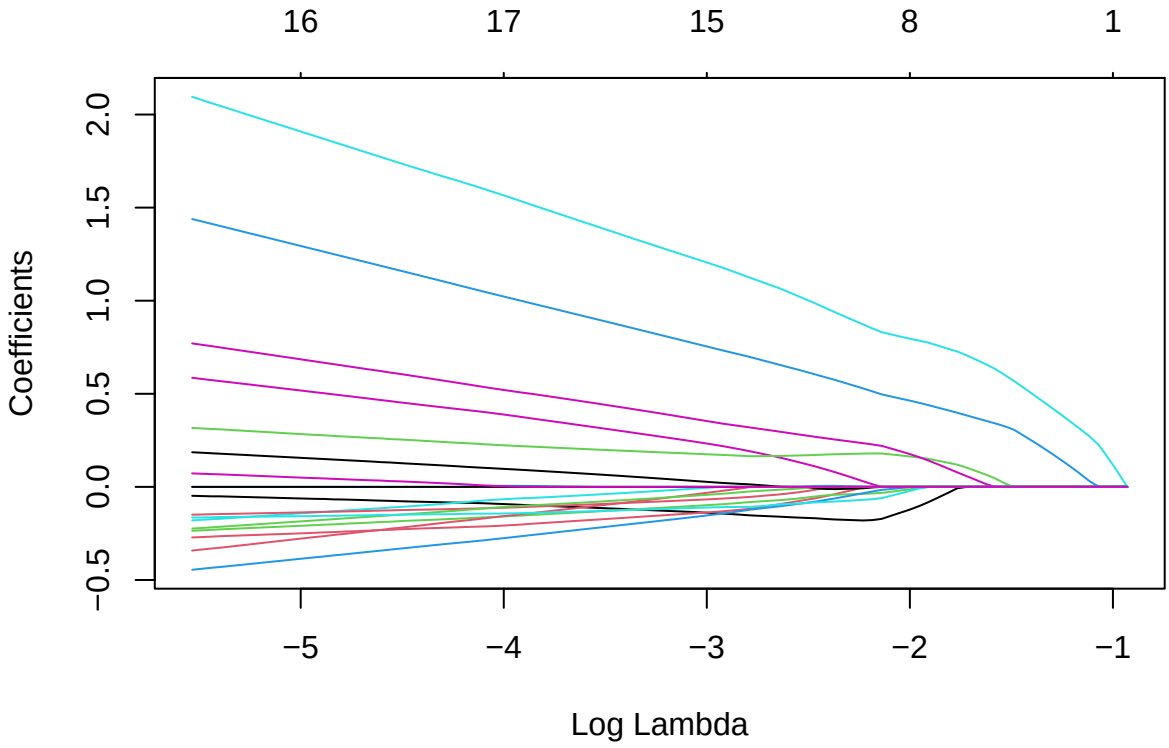
```
model2$ix
```

```
##  [1]  134 1144 1779 1834 2242 3433 4381 4847 4951 5335 6221
```

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

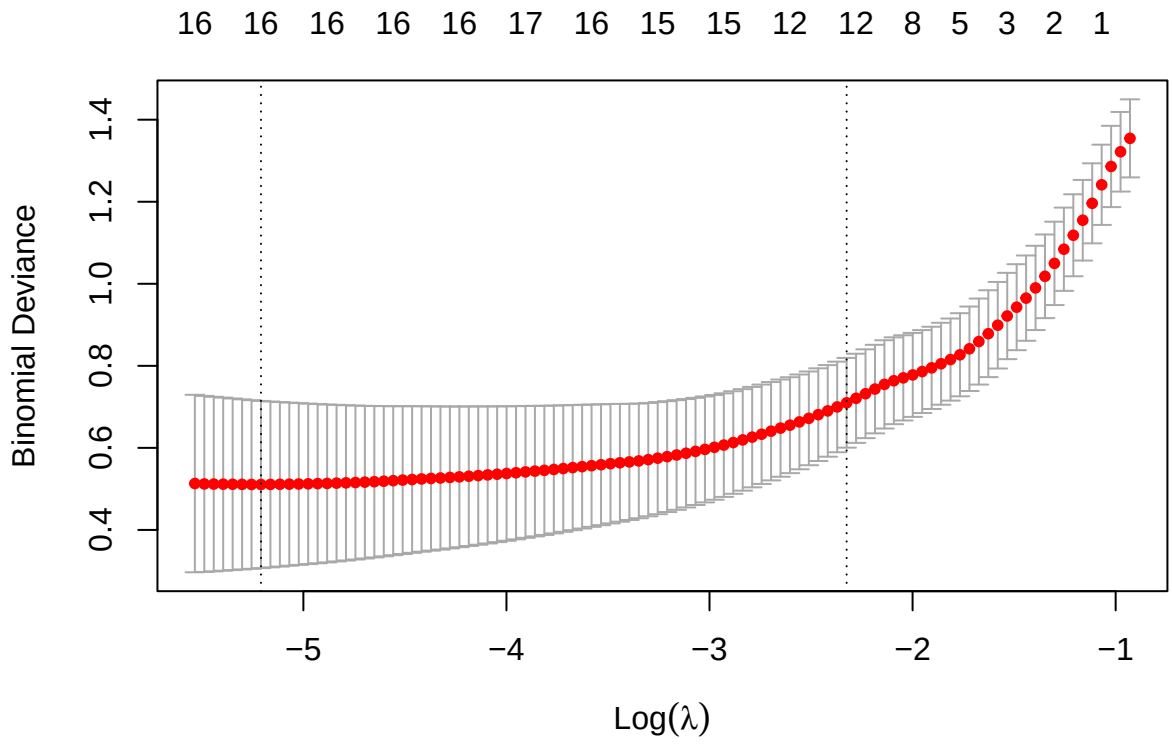
```
modelLasso = glmnet(x.train, y.train, family = "binomial",
                    standardize = F)
```

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



Let's select the tuning parameter:

```
set.seed(1)

modelLassoCV = cv.glmnet(x.train, y.train, family = "binomial",
                         standardize = F,
                         type.measure = "auc")
plot(modelLassoCV)
```




```
modelLassoCV$lambda.min
```

```
## [1] 0.005468224
```

Extract the associated features

```
modelLasso = glmnet(x.train, y.train, family = "binomial",
                    standardize = F,
                    type.measure = "class",
                    lambda=modelLassoCV$lambda.min)
Lassocoef <- which(coef(modelLasso) != 0)

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

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

Let's now compare predictive power:

```
# Make predictions on the testing data
print(paste0("Lasso selected features: ", length(Lassocoef)))
```

```
## [1] "Lasso selected features: 19"
```

```
predLasso <- modelLasso %>% predict(x.test, type='class') %>% as.factor()
confusionMatrix(predLasso, as.factor(y.test))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 24  1
##          1  0 11
```

```
##
##                Accuracy : 0.9722
##                  95% CI : (0.8547, 0.9993)
##     No Information Rate : 0.6667
##     P-Value [Acc > NIR] : 8.699e-06
##
##                   Kappa : 0.9362
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 1.0000
##             Specificity : 0.9167
##          Pos Pred Value : 0.9600
##          Neg Pred Value : 1.0000
##              Prevalence : 0.6667
##          Detection Rate : 0.6667
##    Detection Prevalence : 0.6944
##       Balanced Accuracy : 0.9583
##
##        'Positive' Class : 0
##
```

```
cat("\n\n\n")
```

```
print(paste0("SIS selected features: ", length(model1$ix)))
```

```
## [1] "SIS selected features: 10"
```

```
predSIS <- model1 %>% SIS:::predict.SIS(x.test, type='class') %>% as.factor()
confusionMatrix(predSIS, as.factor(y.test))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 24  2
##          1  0 10
##
##                Accuracy : 0.9444
##                  95% CI : (0.8134, 0.9932)
##     No Information Rate : 0.6667
##     P-Value [Acc > NIR] : 8.081e-05
##
##                   Kappa : 0.8696
##
##  Mcnemar's Test P-Value : 0.4795
##
##             Sensitivity : 1.0000
##             Specificity : 0.8333
##          Pos Pred Value : 0.9231
##          Neg Pred Value : 1.0000
##              Prevalence : 0.6667
##          Detection Rate : 0.6667
##    Detection Prevalence : 0.7222
##       Balanced Accuracy : 0.9167
##
```

```
##        'Positive' Class : 0
##
```

```
cat("\n\n\n")
```

```
print(paste0("ISIS selected features: ", length(model2$ix)))
```

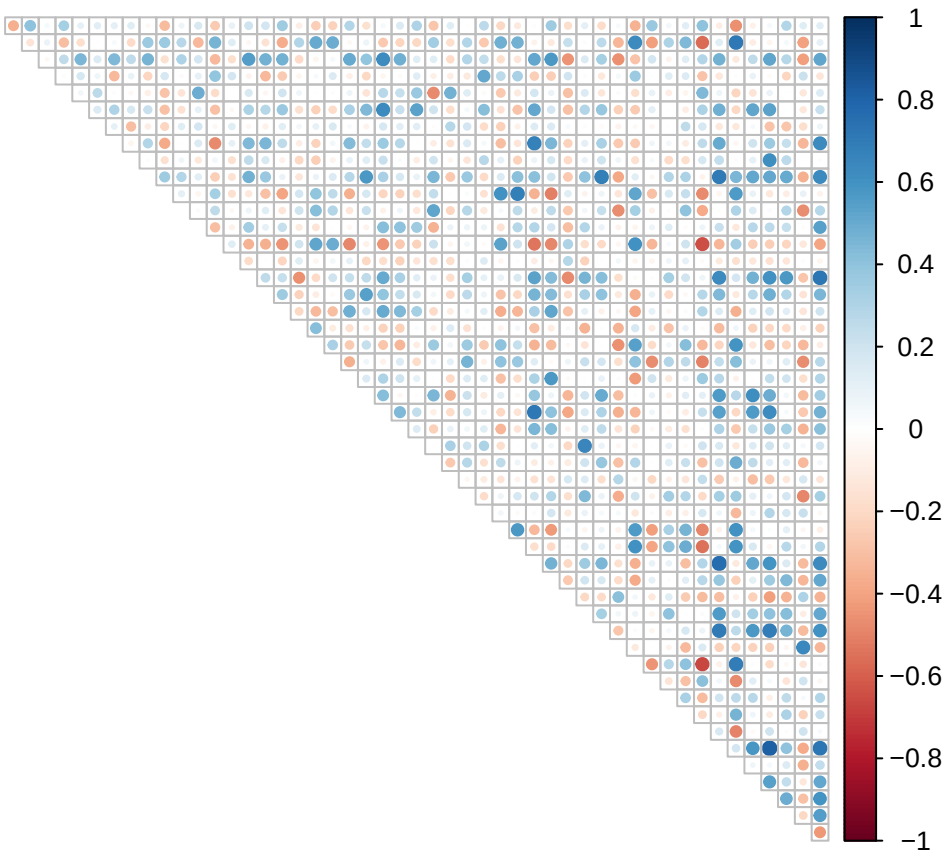
```
## [1] "ISIS selected features: 11"
```

```
predISIS <- model2 %>% SIS:::predict.SIS(x.test, type='class') %>% as.factor()
confusionMatrix(predISIS, as.factor(y.test))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 24  0
##          1  0 12
##
##                Accuracy : 1
##                  95% CI : (0.9026, 1)
##     No Information Rate : 0.6667
##     P-Value [Acc > NIR] : 4.578e-07
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.6667
##          Detection Rate : 0.6667
##    Detection Prevalence : 0.6667
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : 0
##
```

Let's have a look at correlation in the predictors:

```
set.seed(1)
# just take 50 predictors at random
randp <- sample(1:ncol(x.train), 50)
corrplot(cor(x.train)[randp,randp], tl.pos='n', type="upper", diag=F)
```



Note that it is easy to build more involved versions of SIS, such as:

- Var1-ISIS: a first variant of ISIS
- Perm-var-ISIS: permutation-based variant with sample splitting
- etc.

```
# # Var1-ISIS
# model3 = SIS(x.train, y.train, family = "binomial", penalty = "lasso",
#              tune = "cv", nfolds = 10, nsis = 100, varISIS = "aggr", seed = 9,
#              standardize = FALSE)
#
# # Perm-var-ISIS
# model4 = SIS(x.train, y.train, family = "binomial", penalty = "lasso",
#              tune = "cv", nfolds = 10, nsis = 100, varISIS = "aggr", perm = TRUE,
#              q = 0.95, seed = 9, standardize = FALSE)


# selected predictors:
# model3$ix
# model4$ix

# predictive power:
# predISISv1 <- model3 %>% SIS:::predict.SIS(x.test, type='class') %>% as.factor()
# confusionMatrix(predISISv1, as.factor(y.test))
#
# predISISperm <- model4 %>% SIS:::predict.SIS(x.test, type='class') %>% as.factor()
# confusionMatrix(predISISperm, as.factor(y.test))
```