High-dimensional inference

Consider the linear model

$$y = \mathcal{N}(1_n\beta_0 + X\beta, \sigma^2 I_n) \tag{1}$$

where

- $\underset{n\times 1}{y} = (y_1, \ldots, y_n)'$ is the response on n observations
- $\underset{n\times p}{X}$ is the (fixed or random) design matrix containing the measurements on p variables
- $\underset{p\times 1}{\beta} = (\beta_1, \ldots, \beta_p)'$ is the vector of coefficients of interest
- β_0^* and σ^2 are nuisance parameters
- $\underset{n\times 1}{1_n} = (1, 1, \ldots, 1)'$ is a vector of ones of length n and $\underset{n\times n}{I_n}$ is the identity matrix
- We can have a low-dimensional setting ($n \geq p$), but we are particularly interrested in the high-dimensional setting ($n < p$)

The *active set* or the set of relevant variables is defined as

$$S = \{j : \beta_j \neq 0, j = 1, \ldots, p\}$$

and it has cardinality $s = \#S$.

The main goal is the construction of confidence intervals and p-values for individual regression parameters β_j, $j = 1, \ldots, p$. In high-dimensional setting, this makes statistical inference very challenging.

Here the main assumption is that the linear model in (1) is correct. This might be rather unrealistic.

Simulated data

We use the following simulation setting:

- $n = 100$
- $p = 200$
- $\beta_0 = 0$, $\sigma^2 = 1$
- $s = 6$
- $X \sim \mathcal{N}(0, \Sigma)$
- Σ block-diagonal matrix

We divide the variables in 3 types: A, B and C. Type-A variables have $\beta_j \neq 0$, with two strong effects $\beta_j = \pm 1$ and four weak effects $\beta_j = \pm 0.5$; type-B and C variables have $\beta_j = 0$. Each of the six type-A variables is correlated ($\rho = 0.5$) with two other type-B variables. The remaining 42 type-C variables are pure noise and independent of all other variables.

Avoiding the selection of type-B predictors is challenging; however, simple approaches still work well for avoiding the selection of type-C predictors

```
set.seed(1)
n <- 100
p <- 200
s <- 6
```

```
A <- matrix(0.5, 3, 3) + 0.5*diag(3)
B <- diag(p-3*6)
library(Matrix)
Sigma = bdiag(A,A,A,A,A,A,B)
R <- chol(Sigma)
X <- as.matrix(matrix(rnorm(n * p), n, p) %*% R)
beta = numeric(6*3)
beta[(0:5)*3+1] <- c(1,-1,0.5,0.5,-0.5,-0.5)
y <- X[,1:(s*3)] %*% beta + rnorm(n)
varType <- vector("character", p)
varType[(0:5)*3+1] <- "A"
varType[c((0:5)*3+2, (0:5)*3+3)] <- "B"
varType[(s*3+1):p] <- "C"
colnames(X) <- paste0("x", 1:p)
yX = data.frame(y,X)
```

Single-split inference

A generic way for deriving p-values in hypotheses testing is given by *sample-splitting inference*, that is splitting the observations with indices $\{1,\ldots,n\}$ in two equal halves denoted by I_1 and I_2, that is, $I_r \subset \{1,\ldots,n\}$, $r=1,2$ with $I_1 \cup I_2 = \{1,\ldots,n\}$ and $I_1 \cap I_2 = \emptyset$.

The idea is to use the first half I_1 for variable selection and the second half I_2 with the reduced set of selected variables (from I_1) for statistical inference in terms of p-values.

```
set.seed(123)
I1 <- as.logical(sample(rep(0:1, each=n/2)))
```

Such a sample-splitting procedure avoids the over-optimism to use the data twice for selection and inference after selection (without taking the effect of selection into account).

Consider a method for variable selection based on the first half of the sample:

$$\hat{S}(I_1) \subset \{1,\ldots,p\}$$

A prime example is the Lasso which selects all the variables whose corresponding estimated regression coefficients are different from zero.

```
library(glmnet)
```

```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-13
```

```
set.seed(123)
fit <- cv.glmnet(X[I1,], y[I1])
hatbeta <- coef(fit, s=fit$lambda.min)[-1]
tapply(hatbeta!=0, varType, sum)
```

```
##  A  B  C
##  6  1 21
```

We then use the second half of the sample I_2 for constructing p-values, based on the selected variables $\hat{S}(I_1)$.

If the cardinality $\#\hat{S}(I_1) \leq n/2$, we can run ordinary least squares estimation using the subsample I_2 and the selected variables $\hat{S}(I_1)$, that is, we regress y_{I_2} on $X_{I_2}^{\hat{S}(I_1)}$ where the sub-indices denote the sample half and the super-index stands for the selected variables, respectively.

```r
XS <- X[!I1, which(hatbeta!=0)]
fit <- lm(y[!I1]~XS)
summary(fit)
```

```
##
## Call:
## lm(formula = y[!I1] ~ XS)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.55363 -0.47409  0.00665  0.55181  1.49000
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.07258    0.19709   0.368 0.716370
## XSx1         1.15844    0.24647   4.700 0.000122 ***
## XSx4        -0.82708    0.21546  -3.839 0.000955 ***
## XSx7         0.71151    0.20611   3.452 0.002387 **
## XSx10        0.62414    0.22348   2.793 0.010903 *
## XSx13       -0.63470    0.22399  -2.834 0.009950 **
## XSx15        0.27174    0.26587   1.022 0.318386
## XSx16       -0.49667    0.20035  -2.479 0.021741 *
## XSx37        0.14498    0.18418   0.787 0.439967
## XSx46        0.05106    0.22318   0.229 0.821262
## XSx47        0.03909    0.25314   0.154 0.878740
## XSx51        0.27951    0.24209   1.155 0.261242
## XSx52       -0.02980    0.20513  -0.145 0.885873
## XSx73        0.26504    0.20624   1.285 0.212746
## XSx78        0.22871    0.24533   0.932 0.361804
## XSx84        0.02642    0.22279   0.119 0.906734
## XSx95       -0.12322    0.17584  -0.701 0.491152
## XSx100       0.24911    0.32254   0.772 0.448522
## XSx109       0.19304    0.18093   1.067 0.298117
## XSx114       0.42826    0.28385   1.509 0.146253
## XSx123      -0.36149    0.35447  -1.020 0.319431
## XSx146      -0.09395    0.22655  -0.415 0.682566
## XSx154       0.29384    0.24479   1.200 0.243367
## XSx155       0.38648    0.19259   2.007 0.057821 .
## XSx156      -0.24688    0.25186  -0.980 0.338131
## XSx159       0.15939    0.22889   0.696 0.493833
## XSx163      -0.21321    0.23612  -0.903 0.376790
## XSx171      -0.15830    0.25228  -0.627 0.537110
## XSx182      -0.22577    0.26844  -0.841 0.409803
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.017 on 21 degrees of freedom
## Multiple R-squared:  0.8898, Adjusted R-squared:  0.7428
## F-statistic: 6.053 on 28 and 21 DF,  p-value: 3.798e-05
```

Thus, from such a procedure, we obtain p-values for testing $H_j : \beta_j = 0$ for $j \in \hat{S}(I_1)$. Moreover, we define (raw) p-values $p_j = 1$ for $j \notin \hat{S}(I_1)$.

```
p.raw = rep(1,p)
var.id <- as.numeric(gsub("XSx", "", rownames(summary(fit)$coef)[-1] ) )
var.id
```

```
##  [1]   1   4   7  10  13  15  16  37  46  47  51  52  73  78  84  95 100
## [18] 109 114 123 146 154 155 156 159 163 171 182
```

```
p.raw[var.id] = summary(fit)$coefficients[-1,4]
tapply(p.raw <= 0.05, varType, sum)
```

```
## A B C
## 6 0 0
```

An interesting feature of such a sample-splitting procedure is the adjustment for multiple testing. For example, if we wish to control the familywise error rate over all considered hypotheses H_j, $j = 1, \ldots, p$, a naive approach would employ a Bonferroni–Holm correction over the p tests. This is not necessary: we only need to control over the considered $\#\hat{S}(I_1)$ tests in I_2.

```
p.holm = p.adjust(p.raw[var.id], "holm")
names(p.holm) = paste("x", var.id, sep="")
round(p.holm,4)
```

```
##     x1     x4     x7    x10    x13    x15    x16    x37    x46    x47
## 0.0034 0.0258 0.0621 0.2617 0.2488 1.0000 0.5000 1.0000 1.0000 1.0000
##    x51    x52    x73    x78    x84    x95   x100   x109   x114   x123
## 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
##   x146   x154   x155   x156   x159   x163   x171   x182
## 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
```

Such corrected p-values control the familywise error rate in multiple testing when assuming the *screening property*

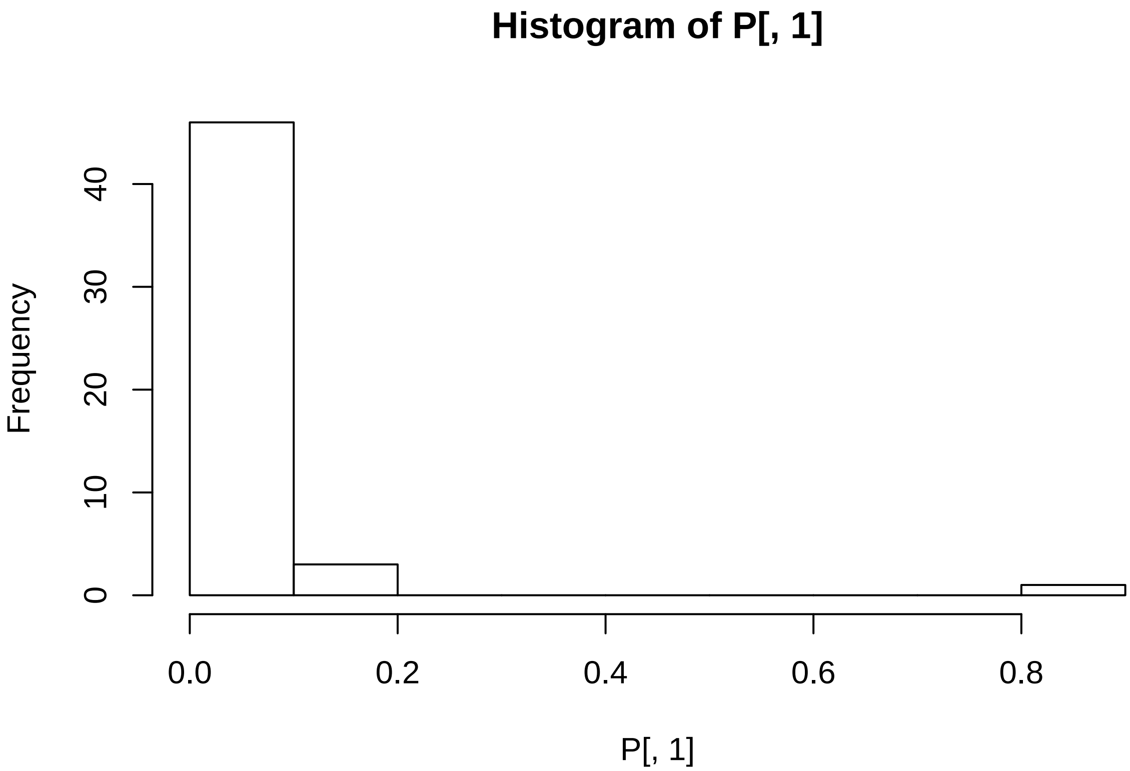
$$\hat{S} = \{j : \hat{\beta}_j \neq 0\} \supseteq S$$

for the selector $\hat{S} = \hat{S}(I_1)$ based on the first half I_1 only. The reason is that the screening property ensures that the reduced model is a correct model, and hence the result is not surprising.

Multiple-split inference

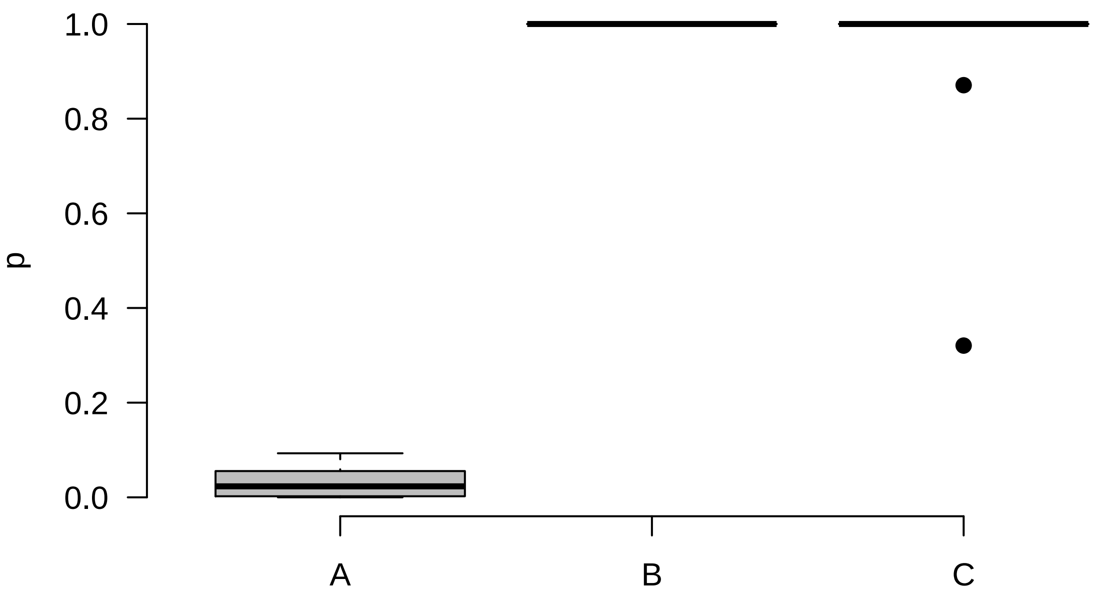
A major problem of the single sample-splitting method is its sensitivity with respect to the choice of splitting the entire sample: sample splits lead to wildly different p-values. We call this undesirable phenomenon a p-value *lottery*, and the following histogram provides an illustration for a single variable.

```
set.seed(123)
B = 50
P <- matrix(1, B, p)
for (i in 1:B) {
  ind <- as.logical(sample(rep(0:1, each=n/2)))
  cvfit <- cv.glmnet(X[ind,], y[ind])
  b <- coef(cvfit, s=cvfit$lambda.min)[-1]
  XX <- X[!ind, which(b!=0)]
  fit <- lm(y[!ind]~XX)
  summ <- summary(fit)$coefficients[-1,]
  var.id <- as.numeric(gsub("XXx", "", rownames(summ)))
  P[i, var.id] <- summ[,4]
}
hist(P[,1])
```



To overcome the p-value lottery, we can run the sample-splitting method B times, with B large.

```
boxplot(apply(P, 2, median)[(0:5)*3+1],
        apply(P, 2, median)[c((0:5)*3+2, (0:5)*3+3)],
        apply(P, 2, median)[(s*3+1):p], col="gray", frame.plot=FALSE, pch=19,
        names=LETTERS[1:3], las=1, ylim=c(0,1), ylab="p")
```



Thus, we obtain a collection of p-values for the jth hypothesis H_j

$$p_j^{[1]}, \ldots, p_j^{[B]}$$

The task is now to do an aggregation to a single p-value. Because of dependence among $\{p_j^{[B]}, b = 1, \ldots, B\}$, because all the different half samples are part of the same full sample, an appropriate aggregation is needed. A simple solution is to use the median of $\{p_j^{[B]}, b = 1, \ldots, B\}$ and multiplying it with the factor 2.

```
p.median = apply(P,2,median)
tapply(p.median <= 0.05, varType, sum)
```

```
## A B C
## 4 0 0
```

The implementation in the R package hdi works as follow

```
library(hdi)
set.seed(123)
fit <- multi.split(x=X, y=y, B=50, fraction=0.5, ci=TRUE, ci.level = 0.95)
fit
```

```
FALSE alpha = 0.01: Selected predictors: 1 4
FALSE alpha = 0.05: Selected predictors: 1 4
FALSE ------
FALSE Familywise error rate controlled at level alpha.
```

To obtain adjusted p-values for H_j controlling the familywise error rate:

```
p.fwer = fit$pval.corr
round(p.fwer,4)
```

```
##     x1     x2     x3     x4     x5     x6     x7     x8     x9    x10
## 0.0000 1.0000 1.0000 0.0041 1.0000 1.0000 0.2536 1.0000 1.0000 1.0000
##    x11    x12    x13    x14    x15    x16    x17    x18    x19    x20
## 1.0000 1.0000 1.0000 1.0000 1.0000 0.3275 1.0000 1.0000 1.0000 1.0000
##    x21    x22    x23    x24    x25    x26    x27    x28    x29    x30
## 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
##    x31    x32    x33    x34    x35    x36    x37    x38    x39    x40
## 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
##    x41    x42    x43    x44    x45    x46    x47    x48    x49    x50
## 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
##    x51    x52    x53    x54    x55    x56    x57    x58    x59    x60
## 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
##    x61    x62    x63    x64    x65    x66    x67    x68    x69    x70
## 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
##    x71    x72    x73    x74    x75    x76    x77    x78    x79    x80
## 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
##    x81    x82    x83    x84    x85    x86    x87    x88    x89    x90
## 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
##    x91    x92    x93    x94    x95    x96    x97    x98    x99   x100
## 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
##   x101   x102   x103   x104   x105   x106   x107   x108   x109   x110
## 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
##   x111   x112   x113   x114   x115   x116   x117   x118   x119   x120
## 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
##   x121   x122   x123   x124   x125   x126   x127   x128   x129   x130
## 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
##   x131   x132   x133   x134   x135   x136   x137   x138   x139   x140
## 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
##   x141   x142   x143   x144   x145   x146   x147   x148   x149   x150
## 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
##   x151   x152   x153   x154   x155   x156   x157   x158   x159   x160
```

```
## 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
##   x161   x162   x163   x164   x165   x166   x167   x168   x169   x170
## 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
##   x171   x172   x173   x174   x175   x176   x177   x178   x179   x180
## 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
##   x181   x182   x183   x184   x185   x186   x187   x188   x189   x190
## 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
##   x191   x192   x193   x194   x195   x196   x197   x198   x199   x200
## 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
```

Confidence intervals can be constructed based on the duality with the p-values:

```
confint(fit, parm=which(p.fwer <= 0.05), level=0.95)
```

```
##         lower      upper
## x1  0.6895478  1.6007416
## x4 -1.1946307 -0.2665034
```