An Introduction to `subDebiased`

Waverly Wei

Jan.2021

Contents

Introduction

subDebiased is a package that implements two bootstrap-assisted estimators: bootstrap-assisted desparsified Lasso and R-split. The two methods remove the subgroup selection bias and regularization bias indeced by high-dimensional covariates.

Installation

```
devtools::install_github("WaverlyWei/subDebiased")
```

Quick start

First we load the `subDebiased` package:

```
library(subDebiased)
```

We generate high-dimensional data with 2 subgroups of interest. We predefine a set of tuning parameters, denoted as r.

```
  library(MASS)

  p <- 200 # number of confounders

  n <- 100 # sample size

  ngroups <- 2 # number of subgroups/treatments;

  s0 <- 4

  m <- ngroups

  Sigma <- matrix(0,p,p)

  for (i in 1:n){
```

```
    for(j in 1:p){
      Sigma[i,j] <- 0.5^(abs(i-j))
    }
  }

  # generate X
  X <- mvrnorm( n = n, mu = rep(0,p), Sigma = Sigma )

  Z <- matrix(0,n,m)

  for(i in 1:n){
    for(j in 1:m){
      Z[i,j] <- rbinom(1,1,exp(X[i,2*j-1] + X[i,2*j])/(1+exp(X[i,2*j-1] + X[i,2*j])))
    }
  }


  # noise: heter/homo
  noise.y <- 1

  betas <- 1

  #index of the subgroups
  w.index <- seq(1, m, 1)

  x <- cbind(Z,X)


  # Model: Y = Z * beta + X * gamma + noise

  # Generate coefficients
  beta <- c(rep(0,m-1),betas)

  gamma <- c(rep(1, s0), rep(0, p-s0))

  beta0 <- c(beta, gamma)

  # Generate noise
  noise <- mvrnorm( n = 1, mu = rep(0,n), Sigma = diag(n) * noise.y )

  # Generate response Y
  Y <- 0.5 + x %*% beta0 + noise

  ## parameters in the function
  r = 1/(3*1:10)
```

Bootstrap-calibrated Desparsified Lasso

Bootstrap iteration is recommended to be B = 200. Here we use B = 5 as a demo.

```
desparse_res <- BSDesparseLasso(y = Y,
                                x = x,
                                r = r,
                                G = w.index,
```

```
                                B = 5)
```

Result summary

Results at index 1 to length(r) correspond to each tuning parameter. Result at index [length(r)+1] corresponds to the simultaneous estimator. Result at index [length(r)+2] corresponds to the optimal tuning. The tuning parameter is selected through `cvDesparse`

```
opt_idx <- length(r) + 2 # optimal result index

desparse_res[[opt_idx]] # extract optimal results
```

```
## $LowerBound
##       95%
## 0.5304324
##
## $UpperBound
##      95%
## 1.385205
##
## $betaMax
## [1] 0.716925
##
## $betaEst
## [1] 0.2960697 0.9578189
##
## $op
## [1] 1
```

Bootstrap-calibrated R-Split

Bootstrap iteration is recommended to be B = 200, BB = 1000. Here we use B = 5 and BB = 10 as a demo. The tuning parameter is selected through `cvSplit`

```
rsplit_res <- BSSplitLasso(y = Y,
                           x = x,
                           r = r,
                           G = w.index,B = 5, BB = 10)
```

```
## Warning in t(sweep(Ycount, 2, apply(Ycount, 2, mean))) * (alphaEst -
## mean(alphaEst)): longer object length is not a multiple of shorter object length

## Warning in t(sweep(Ycount, 2, apply(Ycount, 2, mean))) * (alphaEst -
## mean(alphaEst)): longer object length is not a multiple of shorter object length
```

```
opt_idx <- length(r) + 2 # optimal result index

rsplit_res[[opt_idx]] # extract optimal results
```

```
## $LowerBound
##        95%
## 0.08035282
##
## $UpperBound
##      95%
## 1.082815
```

```
##
## $betaMax
## [1] 0.3823701
##
## $betaEst
## [1] -0.1061913  0.5815840
##
## $modelSize
##  [1] 6 4 5 6 4 6 6 6 6 6
##
## $op
## [1] 1
```