Model-X knockoffs (ASDP)

Yang Meng

9/11/2019

1. Abstract

The report simulated gaussian data with p>n, then constructed Model-X knockoffs by ASDP, equicorrelated method and exact construction, and then compared FDP with the package "knockoff". The result of my code and the package are almost the same, and the difference is caused by the choice of W.

2. Data Simulation

The data matrix X is $p = 500$ by $n = 200$, and only $k = 10$ variables(randomly picked) are non-null. We set the normalized covariance matrix of X to be Σ, and $y = X\beta + \epsilon$.

```
#Simulating data X and y
set.seed(0)
p=500; n=200; k=10
rho=0.4; amplitude=3
Sigma = toeplitz(rho^(0:(p-1)))
X = matrix(rnorm(n*p),n) %*% chol(Sigma)
nonzero = sample(p, k)
beta = amplitude * (1:p %in% nonzero)
y = X %*% beta + rnorm(n)
```

3. Package "knockoff" Approach

We run the "knockoff.filter" function in package "knockoff", with default q=0.1 and "asdp" method. Then we get the result as below.

```
#knockoffs construction by package using ASDP
fdp = function(selected) sum(beta[selected]==0) / max(1, length(selected))
knockoffs = function(X) create.second_order(X, method='asdp')
result = knockoff.filter(X, y, knockoffs=knockoffs)
#variables selected by package using ASDP
print(result$selected)
```

```
##  [1]  12 110 131 238 269 289 336 363 431 451 493
```

```
#actual non-null variables
print(sort(nonzero))
```

```
##  [1]  12 110 131 238 269 289 336 363 431 493
```

```
#FDP by package using ASDP
fdp(result$selected)
```

```
## [1] 0.09090909
```

4. ASDP Construction

In this part, I solved the semidefinite program by using "create.solve_asdp" function to get appropriate s_j's by ASDP. I can also use package "Rdsdp" to solve the semidefinite program.

We set $W_j = |\hat{b}_j(\lambda)| - |\hat{b}_{j+p}(\lambda)|$ here, where λ is chosen by cross validation with one standard error rule.

```
#my knockoff construction using ASDP
set.seed(0)
s.asdp = create.solve_asdp(Sigma)
mu = X - X %*% solve(Sigma) %*% diag(s.asdp)
V = 2 * diag(s.asdp) - diag(s.asdp) %*% solve(Sigma) %*% diag(s.asdp)
X.t = matrix(NA, nrow=n, ncol=p)
for(i in 1:n) X.t[i,] = mvrnorm(n=1, mu[i,], V)
```

Now we compare the result with the "knockoff" package. The result of FDP and variable selection are similar, and the difference may be caused by the choice of W.

```
#comparison to the package
set.seed(0)
XX = cbind(X, X.t)
fit = cv.glmnet(XX, y, nlambda=100, alpha=1, standardize=T)
b = as.vector(coef(fit, s=fit$lambda.1se))[-1]
w = abs(b[1:p]) - abs(b[(p+1):(2*p)])
threshold = function(w, q=.1){
  for (t in sort(abs(w[w!=0]))){
    if (((1 + sum(w<=-t)) / sum(w>=t)) <= q) return(t)
  }}
myresult = which(w>=threshold(w))

#FDP by package using ASDP
fdp(result$selected)
```

```
## [1] 0.09090909
```

```
#my FDP
fdp(myresult)
```

```
## [1] 0
```

```
#variables selected by package using ASDP
print(result$selected)
```

```
##  [1]  12 110 131 238 269 289 336 363 431 451 493
```

```
#my variables selected
print(myresult)
```

```
##  [1]  12 110 131 238 269 289 336 363 431 493
```

```
#actual non-null variables
print(sort(nonzero))
```

```
##  [1]  12 110 131 238 269 289 336 363 431 493
```

5. Equicorrelated Construction

I also tried equicorrelated construction for knockoffs, and the similar result is as below. We notice that FDP is significantly higher than ASDP construction, because $\lambda_{min}(\Sigma)$ tends to be extremely small as p gets large.

```
#equi by package
set.seed(0)
equi_knockoffs = function(X) create.second_order(X, method='equi', shrink=T)
res = knockoff.filter(X, y, knockoffs = equi_knockoffs)
#my equi
s = 2 * min(eigen(Sigma)$values)
mu = X - X %*% solve(Sigma) %*% diag(s,p)
V = 2 * diag(s,p) - diag(s,p) %*% solve(Sigma) %*% diag(s,p)
X.t2 = matrix(NA, nrow=n, ncol=p)
for(i in 1:n) X.t2[i,] = mvrnorm(n=1, mu[i,], V)
#my equi vs equi
XX2 = cbind(X, X.t2)
fit = cv.glmnet(XX2, y, nlambda=100, alpha=1, standardize=T)
b = as.vector(coef(fit, s=fit$lambda.1se))[-1]
w = abs(b[1:p]) - abs(b[(p+1):(2*p)])
threshold = function(w, q=.1){
  for (t in sort(abs(w[w!=0]))){
    if (((1 + sum(w<=-t)) / sum(w>=t)) <= q) return(t)
  }}
myresult2 = which(w>=threshold(w))

#FDP by package using equi
fdp(res$selected)
```

```
## [1] 0.2307692
```

```
#my FDP
fdp(myresult2)
```

```
## [1] 0.1666667
```

```
#variables selected by package using equi
print(res$selected)
```

```
##  [1]  12  86 110 131 238 269 289 336 363 426 431 451 493
```

```
#my variables selected
print(myresult2)
```

```
##  [1]  12 110 131 238 269 289 336 363 426 431 451 493
```

```
#actual non-null variables
print(sort(nonzero))
```

```
##  [1]  12 110 131 238 269 289 336 363 431 493
```

6. Exact Constructions

```
#Simulating data X and y
set.seed(0)
p=200; n=100; k=8
```

```
rho=0.4; amplitude=3
Sigma = toeplitz(rho^(0:(p-1)))
X = matrix(rnorm(n*p),n) %*% chol(Sigma)
nonzero = sample(p, k)
beta = amplitude * (1:p %in% nonzero)
y = X %*% beta + rnorm(n)
```

```
library("FinCovRegularization")
Y <- X
YYt <- Y
for(j in 1:p){
  Ypt <- c()
  Yp <- cbind(YYt[,j],YYt[,-j])
  mu <- colMeans(Yp)
  s2 <- hard.thresholding(cov(Yp),.9)
  s2.c <- s2[1,1] - s2[1,-1] %*% solve(s2[-1,-1]) %*% s2[-1,1]
  for (i in 1:n){
    mu.c <- mu[1] + s2[1,-1] %*% solve(s2[-1,-1]) %*% (YYt[i,-1] - mu[-1])
    Ypt <- c(Ypt, rnorm(1,mu.c,s2.c))
  }
  YYt <- cbind(YYt, Ypt)
}
```

```
fit = cv.glmnet(YYt, y, nlambda=100, alpha=1, standardize=T)
b = as.vector(coef(fit, s=fit$lambda.1se))[-1]
w = abs(b[1:p]) - abs(b[(p+1):(2*p)])
threshold = function(w, q=.2){
  for (t in sort(abs(w[w!=0]))){
    if (((1 + sum(w<=-t)) / sum(w>=t)) <= q) return(t)
  }}
myresult2 = which(w>=threshold(w))
#my FDP
fdp(myresult2)
```

```
## [1] 0.2
```

```
#my variables selected
myresult2
```

```
##  [1]   5  66 103 104 105 132 138 189 194 198
```

```
#actual non-null variables
sort(nonzero)
```

```
## [1]  66 104 105 132 138 189 194 198
```