IIS-SQDA Simulation

zxm

Oracle-assisted IIS

1. 生成模拟数据

   - $(\Omega_1)_{ij} = 0.5^{|i-j|}$
   - $\Omega_2 = \Omega_1 + \Omega$, 其中 Ω 为对称的稀疏矩阵，元素为 0，除了

   $$\Omega_{5,5} = \Omega_{25,25} = \Omega_{45,45} = -0.29$$

   $$\Omega_{5,25} = \Omega_{5,45} = \Omega_{25,45} = -0.15$$

   其他三个非零元素由对称性确定。
   - $\Sigma_1 = \Omega_1^{-1}$, $\Sigma_2 = \Omega_2^{-1}$
   - $\delta = (0.6, 0.8, 0, \ldots, 0)^T$, $\mu_1 = \Sigma_1\delta$, $\mu_2 = \mathbf{0}$
   - class 1 服从 $\mathcal{N}(\mu_1, \Sigma_1)$ 分布，class 2 服从 $\mathcal{N}(\mu_2, \Sigma_2)$

```
# dimensionality
p = 100
# generate precision matrices
# Omega1
omega1 = sapply(1:p, function(x) 0.5^abs(x - 1:p))
# Omega2
omega2 = omega1
id = seq(from = 5, to = 45, by = 20)
for(i in id){
        for(j in id){
                omega2[i, j] = omega2[i, j] + ifelse(i==j, -0.29, -0.15)
        }
}
# delta
delta = c(0.6, 0.8, rep(0, p - 2))
# sigma1 and sigma2
sigma1 = solve(omega1); sigma2 = solve(omega2)
# mu1 and mu2
mu1 = sigma1%*%delta; mu2 = rep(0, p)
# sample size
n1 = 100; n2 = 100; n = n1 + n2
```

```r
# data points
library(MASS)
set.seed(2016)
Z1 = mvrnorm(n1, mu = mu1, Sigma = sigma1) # n1 times p data matrix
Z2 = mvrnorm(n2, mu = mu2, Sigma = sigma2) # n2 times p data matrix
Z = rbind(Z1, Z2) # n times p data matrix
```

2. 进行变换并计算检验统计量

```r
# transformation
W = Z%*%omega1; V = Z%*%omega2
# test statistics
fun = function(x){
        log(var(x))-(n1/n)*log(var(x[1:n1]))-(n2/n)*log(var(x[(n1 + 1):n]))
}
D1 = apply(W, 2, fun); D2 = apply(V, 2, fun)
```

对检验统计量进行排序，看一下排名靠前的一些，发现对这样的模拟数据，做了右乘 Ω_1 的变换后，得到的检验统计量前三位对应的下标就是所设置的三个交互变量 (interaction variable)，而排在后面的变量下标排名和右乘 Ω_2 变换后正好是相同的。除了交互变量以外的排名靠前的变量会随着模拟数据而改变，因此设定了种子 2016。

```r
(sort(D1, decreasing = TRUE)[1:10])
```

```
##  [1] 0.66678935 0.65156609 0.54623132 0.28566683 0.11233972 0.05782758
##  [7] 0.04801672 0.03780686 0.03116455 0.02452322
```

```r
(rev(order(D1))[1:10])
```

```
##  [1] 45 25  5  2  1 28 53 52 44 27
```

```r
(sort(D2, decreasing = TRUE)[1:10])
```

```
##  [1] 0.28566683 0.11233972 0.05782758 0.04801672 0.03780686 0.03116455
##  [7] 0.02452322 0.02335943 0.02279823 0.02101309
```

```r
(rev(order(D2))[1:10])
```

```
##  [1]  2  1 28 53 52 44 27 29 92 12
```

还可以从 sort 函数的结果看到，统计量有一个突降的过程，可以作为选定阈值的参考。

3. 降低 Ω 的稀疏性，改变

```
(id = seq(from = 1, to = 70, by = 10))
```

```
## [1]  1 11 21 31 41 51 61
```

并为了保证精度矩阵的可逆性，改变了 Ω 非对角元，其实同时也减小了信号。

```
for(i in id){
        for(j in id){
                omega2[i, j] = omega2[i, j] + ifelse(i==j, -0.29, -0.05)
        }
}
```

最后得到的结果中看到，交互变量也会排在前面，只是没有之前的结果那么好了。sort 函数的结果可以看出，差异也没有那么大了。

```
(sort(D1, decreasing = TRUE)[1:10])
```

```
##  [1] 0.28389031 0.26982481 0.24030058 0.21917340 0.21121346 0.18215322
##  [7] 0.11481607 0.08250370 0.04530052 0.03552886
```

```
(rev(order(D1))[1:10])
```

```
##  [1] 31  1 41 21 61  2 11 51 16 90
```

```
(sort(D2, decreasing = TRUE)[1:10])
```

```
##  [1] 0.18215322 0.12021398 0.04530052 0.03552886 0.03429771 0.03341013
##  [7] 0.03178764 0.02675483 0.02633685 0.02508228
```

```
(rev(order(D2))[1:10])
```

```
##  [1]  2  1 16 90 53 52 59 68 41 43
```

Unknown precision matrices

精度矩阵的估计采取的是 CLIME，由于 `clime` 这个包跑起来速度很慢，以下采用 $p = 50$。

```
# dimensionality
p = 50
# generate precision matrices
# Omega1
omega1 = sapply(1:p, function(x) 0.5^abs(x - 1:p))
# Omega2
omega2 = omega1
id = seq(from = 5, to = 45, by = 20)
for(i in id){
        for(j in id){
                omega2[i, j] = omega2[i, j] + ifelse(i==j, -0.29, -0.15)
        }
}
# delta
delta = c(0.6, 0.8, rep(0, p - 2))
# sigma1 and sigma2
sigma1 = solve(omega1); sigma2 = solve(omega2)
# mu1 and mu2
mu1 = sigma1%*%delta; mu2 = rep(0, p)
# sample size
n1 = 100; n2 = 100; n = n1 + n2
# data points
library(MASS)
set.seed(2016)
Z1 = mvrnorm(n1, mu = mu1, Sigma = sigma1) # n1 times p data matrix
Z2 = mvrnorm(n2, mu = mu2, Sigma = sigma2) # n2 times p data matrix
Z = rbind(Z1, Z2) # n times p data matrix

# CLIME
library(clime)
```

```
## Loading required package: lpSolve
```

```
re.clime = clime(Z1, standardize = FALSE, linsolver = "simplex")
re.cv = cv.clime(re.clime)
re.clime.opt = clime(Z1, standardize = FALSE, re.cv$lambdaopt)
```

```r
hatomega1 = re.clime.opt$Omegalist[[1]]

re.clime = clime(Z2, standardize = FALSE, linsolver = "simplex")
re.cv = cv.clime(re.clime)
re.clime.opt = clime(Z2, standardize = FALSE, re.cv$lambdaopt)
hatomega2 = re.clime.opt$Omegalist[[1]]

# transformation
W = Z%*%hatomega1; V = Z%*%hatomega2

# test statistics
fun = function(x){
        log(var(x))-(n1/n)*log(var(x[1:n1]))-(n2/n)*log(var(x[(n1 + 1):n]))
}
D1 = apply(W, 2, fun); D2 = apply(V, 2, fun)

(sort(D1, decreasing = TRUE)[1:10])
```

```
##  [1] 1.00919152 0.92990190 0.90613982 0.27239134 0.15819476 0.15488756
##  [7] 0.09114001 0.08817124 0.06787483 0.06748810
```

```r
(rev(order(D1))[1:10])
```

```
##  [1] 25  5 45  2 26 46  1  3 36 43
```

```r
(sort(D2, decreasing = TRUE)[1:10])
```

```
##  [1] 0.25676984 0.09006619 0.07943845 0.07557112 0.06421415 0.05942293
##  [7] 0.05537023 0.04871783 0.04564021 0.04037138
```

```r
(rev(order(D2))[1:10])
```

```
##  [1]  2 33  1 44 38 14 39 13 29 11
```

由于需要使用 `clime` 包来估计精度矩阵的估计 $\hat{\Omega}_1$ 和 $\hat{\Omega}_2$，结果相对 oracle 的情况会增加一些随机性的误差。

另一种可行的做法就是直接对 $\hat{\Omega}_2 - \hat{\Omega}_1$ 所有元素的绝对值排序来筛选出那些交互项。

```r
diff = abs(as.vector(hatomega2-hatomega1))
sort(diff, decreasing = TRUE)[1:50]
```

```
##  [1] 0.8861180 0.7774648 0.7459262 0.7133376 0.6945568 0.6703628 0.6700485
##  [8] 0.6569120 0.6468923 0.6229576 0.6109313 0.5836777 0.5488230 0.5464050
## [15] 0.5452504 0.5438496 0.5301508 0.5294984 0.5149189 0.5090343 0.4959279
## [22] 0.4925263 0.4883631 0.4878759 0.4686717 0.4680903 0.4623514 0.4605704
## [29] 0.4592180 0.4476866 0.4388515 0.4365272 0.4280223 0.4279826 0.4247291
## [36] 0.4219742 0.4193506 0.4189474 0.4185214 0.4057639 0.4013598 0.3974406
## [43] 0.3971929 0.3940489 0.3773025 0.3745471 0.3531271 0.3531271 0.3471175
## [50] 0.3458904
```

```r
result = rev(order(diff))[1:50]
paste(ceiling(result/50), result%%50)
```

```
##  [1] "22 22" "46 46" "2 2"   "5 5"   "31 31" "25 25" "50 0"  "45 45"
##  [9] "28 28" "36 36" "1 1"   "6 6"   "18 18" "30 30" "43 43" "21 21"
## [17] "42 42" "7 7"   "15 15" "35 35" "23 23" "12 12" "9 9"   "37 37"
## [25] "47 47" "39 39" "3 3"   "13 13" "8 8"   "40 40" "24 24" "11 11"
## [33] "32 32" "41 41" "49 49" "34 34" "44 44" "16 16" "48 48" "20 20"
## [41] "26 26" "4 4"   "27 27" "19 19" "38 38" "29 29" "22 21" "21 22"
## [49] "10 10" "17 17"
```

```r
result2 = rev(order(diff))[51:60]
paste(ceiling(result2/50), result2%%50)
```

```
##  [1] "14 14" "7 6"   "6 7"   "47 46" "46 47" "44 43" "43 44" "9 8"
##  [9] "8 9"   "39 38"
```

从以上的结果可以看到，效果不如 IIS，但是模型中起作用的平方项大致都排在前面，所以这种方法也是可行的。同时看到两个精度矩阵元素之差的绝对值排在前 50 的 (p=50) 的都是对角元，原因应该是 CLIME 对对角元的估计比较差。

```r
# error rank of CLIME
error = rev(order(abs(as.vector(omega1-hatomega1))))[1:50]
paste(ceiling(error/50), error%%50)
```

```
##  [1] "33 33" "10 9"  "9 10"  "3 2"   "2 3"   "26 25" "25 26" "34 33"
##  [9] "33 34" "14 14" "14 13" "13 14" "10 10" "17 17" "8 7"   "7 8"
```

```
## [17] "49 48" "48 49" "33 32" "32 33" "17 16" "16 17" "5 4"   "4 5"
## [25] "19 19" "38 38" "29 29" "27 26" "26 27" "26 26" "4 4"   "41 40"
## [33] "40 41" "18 17" "17 18" "27 27" "36 35" "35 36" "37 36" "36 37"
## [41] "20 20" "16 16" "32 31" "31 32" "48 48" "49 49" "40 39" "39 40"
## [49] "24 23" "23 24"
```

```
error2 = rev(order(abs(as.vector(omega2-hatomega2))))[1:50]
paste(ceiling(error2/50), error2%%50)
```

```
##  [1] "46 46" "19 19" "28 28" "26 26" "49 49" "16 16" "18 18" "24 24"
##  [9] "3 3"   "48 48" "4 4"   "7 7"   "37 37" "6 6"   "31 31" "32 32"
## [17] "11 11" "43 43" "8 8"   "22 22" "10 10" "20 20" "34 34" "30 30"
## [25] "21 21" "35 35" "27 27" "17 17" "23 23" "36 36" "12 12" "15 15"
## [33] "42 42" "13 13" "41 41" "40 40" "38 38" "47 47" "44 44" "9 9"
## [41] "33 33" "29 29" "2 2"   "50 0"  "39 39" "14 14" "1 1"   "5 5"
## [49] "45 45" "25 25"
```

```
error3 = rev(order(abs(as.vector(omega2 - omega1 - hatomega2 + hatomega1))))[1:50]
paste(ceiling(error3/50), error3%%50)
```

```
##  [1] "22 22" "46 46" "2 2"   "31 31" "50 0"  "28 28" "36 36" "1 1"
##  [9] "6 6"   "18 18" "30 30" "43 43" "21 21" "42 42" "7 7"   "15 15"
## [17] "35 35" "23 23" "12 12" "9 9"   "37 37" "47 47" "39 39" "3 3"
## [25] "13 13" "8 8"   "40 40" "24 24" "11 11" "32 32" "41 41" "49 49"
## [33] "5 5"   "34 34" "44 44" "16 16" "48 48" "20 20" "26 26" "4 4"
## [41] "27 27" "19 19" "25 25" "38 38" "29 29" "45 45" "22 21" "21 22"
## [49] "10 10" "17 17"
```

而两个不同特征相乘的交互项，即使是真正起作用的，也排在非常靠后的位置 (这次模拟 2500 位中排在 200 位左右)，因此这么做其实和 IIS 一样都只能筛选出那些交互变量，而不是最后的交互项。如果一个真实模型只有交互项中只有交叉项起作用，而平方项不起作用，那这个方法和 IIS 一样筛选不出起作用的那些交互项。IIS 提供了选择阈值的理论，并且从 `sort` 函数的结果中也可以看出，IIS 的统计量会有一个突降的过程，可以以此作为选定阈值的依据。而第二种做法 `sort` 函数的结果连续变化，差异不大，难以选定好阈值或者特征个数。