MM-Ridge-test

Matias Salibian

September 13, 2017

MM-Ridge

Below is a comparison (aka “sanity check”) between the implementations of the S-ridge estimators in packages `mmlasso` and `pense`. In addition, I also look at the output of the MM-ridge estimator computed with `pense::mstep` setting `alpha = 0`.

We first load the libraries and generate the data:

```
library(pense)
```

```
## Loading required package: Matrix
```

```
library(mmlasso)
# simple synthetic example
n <- 50 # 500
p <- 20 # 10
set.seed(123)
x <- matrix(rnorm(n*p), n, p)
y <- as.vector( x %*% c(rep(2, 5), rep(0, p-5))) + rnorm(n, sd=.5)
```

We now use `mmlasso::sridge` and `pense::pense` (the latter with `alpha= 0`) to compute an S-ridge estimator.

```
# mmlasso S-ridge
a <- sridge(x=x, y=y, cualcv.S=5, numlam.S=30, niter.S=50, normin=0,
            denormout=0, alone=1, ncores=4)
# pense S-ridge
b0 <- pense(X=x, y=y, alpha=0, standardize=TRUE, lambda=1e-9, initial='cold',
           options=pense_options(delta=a$delta))
```

Note that the optimal value of the penalization found by `mmlasso::sridge` is 0 but `pense::pense()` does not accept `lambda=0` as an argument, so I used `lambda = 1e-9` above.

Also note that I did set `options=pense_options(delta=a$delta)` above to make sure `pense::pense()` was optimizing the same M-scale as `mmlasso::sridge()`. This value is adjusted internally, and for this example it was equal to 0.3.

Although the estimated residual scales are somewhat different (0.3157082, 0.2379258, for `sridge` and `pense`, respectively), the regression estimators are similar:

```
cbind(a$coef, as.vector(b0$coef[,1]))
```

```
##               [,1]        [,2]
##  [1,] -0.060788408 -0.04788235
##  [2,]  1.986060926  1.97005612
##  [3,]  1.949171337  1.95071591
##  [4,]  1.884465500  1.88718058
##  [5,]  2.100334127  2.09235680
##  [6,]  1.946810181  1.94531158
##  [7,]  0.126814576  0.12918098
##  [8,]  0.037691818  0.03975092
##  [9,] -0.068195856 -0.07363655
## [10,]  0.147353461  0.14684796
## [11,]  0.081355418  0.07122246
## [12,]  0.009756274  0.01549973
## [13,]  0.203525559  0.22044957
## [14,]  0.029724836  0.02748962
## [15,] -0.016256748 -0.02014535
## [16,]  0.042139038  0.03303540
## [17,] -0.179635198 -0.17055090
## [18,] -0.146294321 -0.15196289
## [19,]  0.128696020  0.13239447
## [20,]  0.323247188  0.33287999
## [21,]  0.066631848  0.06680060
```

We can now use `pense::mstep()` to do the M-step starting from the S-ridge estimator as computed by `pense::pense()`:

```
g <- mstep(b0, complete_grid=TRUE)
cbind(a$coef, as.vector(b0$coef[,1]), g$coefficients[,1])
```

```
##                     [,1]        [,2]         [,3]
## (Intercept) -0.060788408 -0.04788235 -0.123863757
## X1           1.986060926  1.97005612  2.150531548
## X2           1.949171337  1.95071591  1.968262313
## X3           1.884465500  1.88718058  1.919214193
## X4           2.100334127  2.09235680  2.194127049
## X5           1.946810181  1.94531158  2.008608122
## X6           0.126814576  0.12918098 -0.013253010
## X7           0.037691818  0.03975092  0.045042565
## X8          -0.068195856 -0.07363655 -0.030294561
## X9           0.147353461  0.14684796  0.055192947
## X10          0.081355418  0.07122246 -0.025809588
## X11          0.009756274  0.01549973  0.186318038
## X12          0.203525559  0.22044957  0.020032766
## X13          0.029724836  0.02748962  0.073339909
## X14         -0.016256748 -0.02014535 -0.093097116
## X15          0.042139038  0.03303540  0.228982879
## X16         -0.179635198 -0.17055090 -0.106136525
## X17         -0.146294321 -0.15196289 -0.103611413
## X18          0.128696020  0.13239447  0.001414693
## X19          0.323247188  0.33287999  0.175339742
## X20          0.066631848  0.06680060 -0.184113037
```

This looks reasonable, however the scale estimators can be a bit different:

```
c(a$scale, b0$scale, g$scale)
```

```
##               scale     scale
## 0.3157082 0.2379258 0.4676508
```

Just for the record, the optimal value of the penalization found in this M-step was

```
g$lambda
```

```
## [1] 0.0008636253
```