MM-Ridge-test

Matias Salibian-Barrera

September 13, 2017

Comparing S and MM-Ridge implementations

Below I compare the implementations of the S-ridge estimator that are present in the packages `mmlasso` and `pense`. In addition I also look at the output of the MM-ridge estimator computed with the function `pense::mstep()` setting `alpha = 0`.

TL;DR

The main conclusions seem to be that

- `mmlasso::sridge()` and `pense::pense()` behave similarly enough (at least when `alpha = 0`), but are not identical, and can be rather different (see the repetition of the example below but with `n = 500` instead, for example). This difference does not seem to dissappear with larger samples.
- the residual scale returned by `mmlasso::sridge()` can be rather different from the one in `pense::pense()` (**Can this have an effect on the resulting PENSE-M we use in our simulations?**); and
- the function `pense::mstep()` returns yet another residual scale estimate, this doesn't seem to be documented. Furthermore, for `n = 500` this difference seems to dissappear.

S-ridge in packages mmlasso and pense

We first load the libraries and generate a simple synthetic data set:

```r
library(pense)
library(mmlasso)
# simple synthetic example
n <- 100
p <- 20
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.

```r
# 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 0.4.

Although the estimated residual scales are somewhat different (0.3625265, 0.3094789, for `sridge` and `pense`, respectively), the regression estimators are similar:

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

```
##              [,1]        [,2]
##  [1,]  0.01269419  0.01271584
##  [2,]  1.97032229  1.97029836
##  [3,]  1.91359916  1.91360028
##  [4,]  1.96329897  1.96330465
##  [5,]  2.11922970  2.11922562
##  [6,]  2.06746205  2.06747459
##  [7,]  0.14293069  0.14293506
##  [8,]  0.03526848  0.03527521
##  [9,] -0.08493850 -0.08492049
## [10,]  0.02406280  0.02405967
## [11,] -0.06284134 -0.06284057
## [12,] -0.02720469 -0.02721590
## [13,]  0.08168929  0.08169467
## [14,] -0.17589025 -0.17593158
## [15,] -0.03618143 -0.03617467
## [16,]  0.04700106  0.04701667
## [17,] -0.10778835 -0.10778037
## [18,] -0.06773025 -0.06773179
## [19,]  0.02270311  0.02269785
## [20,]  0.06384142  0.06385664
## [21,] -0.06126596 -0.06127310
```

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)  0.01269419  0.01271584 -0.0555145458
## X1           1.97032229  1.97029836  1.9469089802
## X2           1.91359916  1.91360028  1.9525217364
## X3           1.96329897  1.96330465  2.0350277726
## X4           2.11922970  2.11922562  1.9972698115
## X5           2.06746205  2.06747459  2.0113065856
## X6           0.14293069  0.14293506  0.0054006059
## X7           0.03526848  0.03527521  0.0301580223
## X8          -0.08493850 -0.08492049 -0.0706045624
## X9           0.02406280  0.02405967  0.0003446244
## X10         -0.06284134 -0.06284057 -0.0096677443
## X11         -0.02720469 -0.02721590  0.0646317325
## X12          0.08168929  0.08169467  0.0168366767
## X13         -0.17589025 -0.17593158 -0.0569244618
## X14         -0.03618143 -0.03617467  0.0009537604
## X15          0.04700106  0.04701667  0.0324032644
## X16         -0.10778835 -0.10778037 -0.0340894661
## X17         -0.06773025 -0.06773179 -0.0388196564
## X18          0.02270311  0.02269785 -0.0052466083
## X19          0.06384142  0.06385664  0.0185000188
## X20         -0.06126596 -0.06127310 -0.0827143583
```

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

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

```
##                scale     scale
## 0.3625265 0.3094789 0.4172325
```

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

```
g$lambda
```

```
## [1] 0.006291456
```

Nevertheless, things can be rather different sometimes

If we repeat the same experiment as above, but with `n = 50` instead of `n = 100` we see that the two implementations of S-ridge can be rather different:

```
# another simple synthetic example
n <- 50
p <- 20
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)
# 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))
```

The optimal penalization from `sridge` is still 0. The scales and regression coefficients:

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

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

```
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
```

Not surprisingly, the difference carries over to the M-step:

```
g <- mstep(b0, complete_grid=TRUE)
c(a$scale, b0$scale, g$scale)
```

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

```
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
```

Something weird for large sample sizes?

If we repeat the same experiment as above, but with `n = 500`, we see that the difference between the `mmlasso` and `pense` implementations of S-ridge remain in place, but intriguingly, the residual scale estimator reported by `pense::mstep` is **now the same as** the one returned by `pense::pense`.

```
# another simple synthetic example
n <- 500
p <- 20
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)
# 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
```

```r
b0 <- pense(X=x, y=y, alpha=0, standardize=TRUE, lambda=1e-9, initial='cold',
            options=pense_options(delta=a$delta))
```

The optimal penalization from `sridge` is still 0. The scales and regression coefficients:

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

```
##                 scale
## 0.4797006 0.4514077
```

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

```
##               [,1]          [,2]
##  [1,]  0.019757187  0.0043116664
##  [2,]  2.041996592  1.9676962832
##  [3,]  1.981168116  1.9474561770
##  [4,]  1.993376870  1.9568488252
##  [5,]  2.002878613  2.0008489044
##  [6,]  2.020984266  1.9283820898
##  [7,] -0.057560731 -0.0872649218
##  [8,]  0.003778828  0.0147837873
##  [9,]  0.026307565  0.0088495568
## [10,] -0.007791711  0.0321157244
## [11,] -0.041027701  0.0302009437
## [12,] -0.001861261 -0.0196920085
## [13,] -0.039223236 -0.0706552946
## [14,] -0.012318202 -0.0573767985
## [15,] -0.036412483  0.0045470542
## [16,]  0.051411425  0.0422741668
## [17,]  0.008248508 -0.0335936288
## [18,] -0.012816424 -0.0301062345
## [19,]  0.014141842  0.0277944131
## [20,] -0.006519619 -0.0009197938
## [21,]  0.038129072 -0.0072159631
```

Not surprisingly, the difference carries over to the M-step:

```r
g <- mstep(b0, complete_grid=TRUE)
c(a$scale, b0$scale, g$scale)
```

```
##                 scale     scale
## 0.4797006 0.4514077 0.4514077
```

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

```
##                     [,1]          [,2]         [,3]
## (Intercept)  0.019757187  0.0043116664  0.025806178
## X1           2.041996592  1.9676962832  1.994772873
## X2           1.981168116  1.9474561770  1.979543070
## X3           1.993376870  1.9568488252  1.995976719
## X4           2.002878613  2.0008489044  2.008804852
## X5           2.020984266  1.9283820898  1.992841906
## X6          -0.057560731 -0.0872649218 -0.038952710
## X7           0.003778828  0.0147837873  0.018408864
## X8           0.026307565  0.0088495568  0.036698958
## X9          -0.007791711  0.0321157244  0.019436707
## X10         -0.041027701  0.0302009437  0.002574119
```

```
## X11          -0.001861261 -0.0196920085 -0.009899735
## X12          -0.039223236 -0.0706552946 -0.012203024
## X13          -0.012318202 -0.0573767985 -0.033507504
## X14          -0.036412483  0.0045470542 -0.021075671
## X15           0.051411425  0.0422741668  0.031235801
## X16           0.008248508 -0.0335936288  0.014224165
## X17          -0.012816424 -0.0301062345 -0.004283735
## X18           0.014141842  0.0277944131  0.019277334
## X19          -0.006519619 -0.0009197938  0.027669778
## X20           0.038129072 -0.0072159631 -0.007132149
```