hw8_yw3204

wyh

11/27/2018

1.

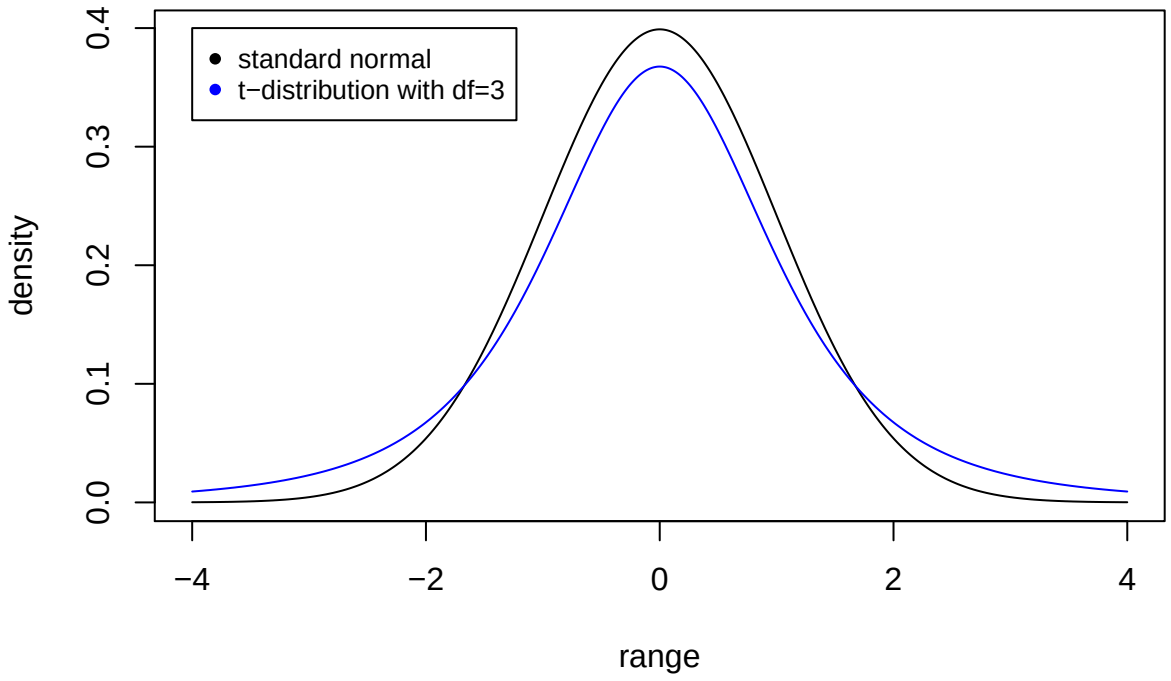
```
n <- 100
p <- 10
s <- 3
set.seed(0)
x <- matrix(rnorm(n*p), n, p)
b <- c(-0.7, 0.7, 1, rep(0, p-s))
y <- x %*% b + rt(n, df=2)

cor(x, y)
```

```
##                [,1]
##  [1,] -0.2526434175
##  [2,]  0.1239284685
##  [3,]  0.1673840288
##  [4,] -0.2522804417
##  [5,] -0.0371161818
##  [6,]  0.1561141420
##  [7,] -0.1175268150
##  [8,] -0.0899681839
##  [9,] -0.0002104895
## [10,]  0.0506851086
```

It is not possible to pick out each of the 3 relevant variables based on correlations alone. Theoreticcaly, the three relevant variables should be the first three. But based on the correlation, the three chosen are apparently not the first three.

2.

```
range <- seq(-4, 4, 0.01)
plot(range, dnorm(range), type = "l", ylab = "density")
lines(range, dt(range, 3), col = "blue")
legend(-4, 0.4, legend=c("standard normal", "t-distribution with df=3"), col=c("black", "blue"), pch=c(
```



3.

```
psi <- function(r, c = 1) {
  return(ifelse(r^2 > c^2, 2*c*abs(r) - c^2, r^2))
}

huber.loss <- function(beta) {
  resd <- x %*% beta - y
  res <- sum(psi(resd))
  return(res)
}
```

4.

```
library(numDeriv)

grad.descent <- function(f, x0, max.iter = 200, step.size = 0.05, stopping.deriv = 0.01, ...) {

  n    <- length(x0)
  xmat <- matrix(0, nrow = n, ncol = max.iter)
  xmat[,1] <- x0

  for (k in 2:max.iter) {
    # Calculate the gradient
    grad.cur <- grad(f, xmat[ ,k-1], ...)

    # Should we stop?
    if (all(abs(grad.cur) < stopping.deriv)) {
      k <- k-1; break
```

```
    }

    # Move in the opposite direction of the grad
    xmat[ ,k] <- xmat[ ,k-1] - step.size * grad.cur
  }

  xmat <- xmat[ ,1:k] # Trim
  return(list(x = xmat[,k], xmat = xmat, k = k))
}

gd <- grad.descent(huber.loss, x0 = rep(0, p), step.size = 0.001, stopping.deriv = 0.1)
gd$x
```

```
##  [1] -0.87346579  0.61828938  0.87989797 -0.04910821  0.07277491
##  [6]  0.10229815 -0.12513246 -0.14559243 -0.11903666 -0.02250130
```

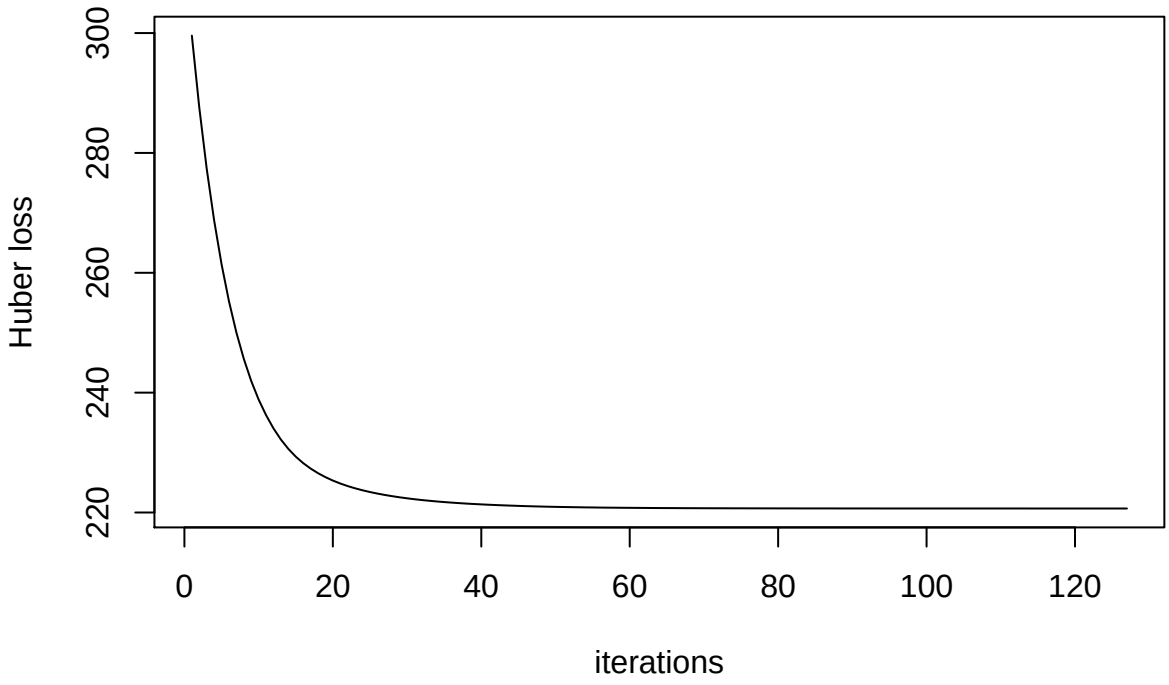
```
gd$k
```

```
## [1] 127
```

The final coefficients are listed above and it takes 127 steps to converge.

5.

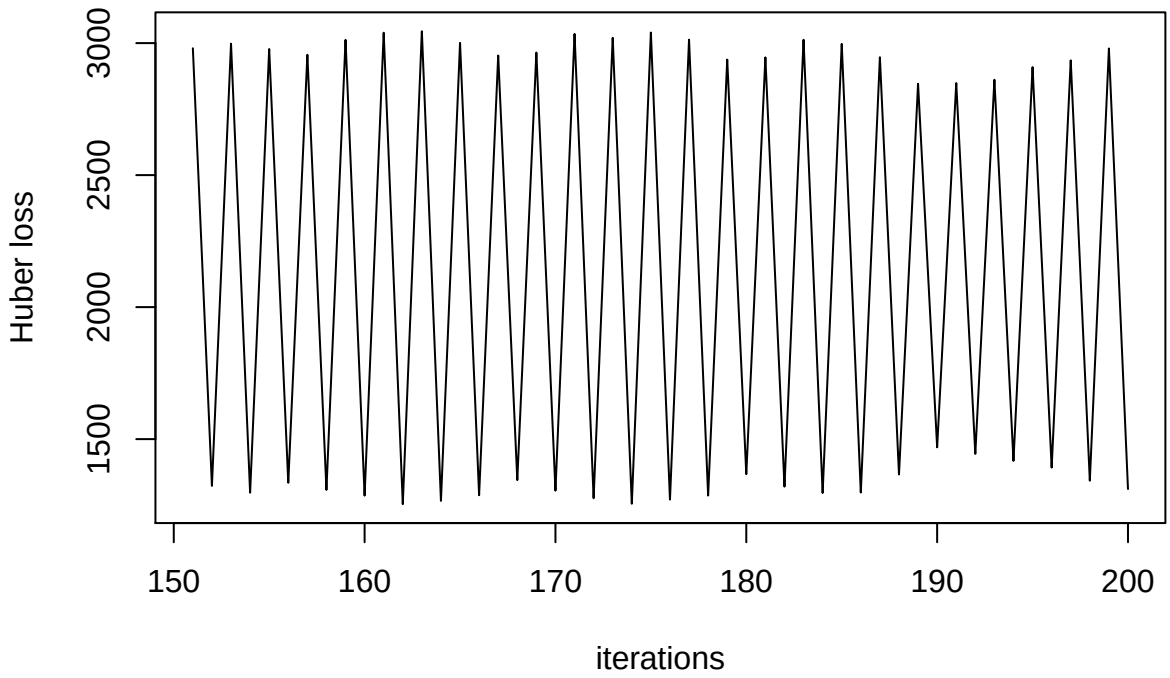
```
obj <- apply(gd$xmat, 2, huber.loss)

plot(1:127, obj, type = "l", xlab = "iterations", ylab = "Huber loss")
```



At start, the algorithm converges quickly and when it is near the minimum, it converges slowly.

6.

```
gd1 <- grad.descent(huber.loss, x0 = rep(0, p), step.size = 0.1, stopping.deriv = 0.1)

obj1 <- apply(gd1$xmat, 2, huber.loss)

#plot(1:200, obj1[1:200], type = "l", xlab = "iterations", ylab = "Huber loss")
plot(151:200, obj1[151:200], type = "l", xlab = "iterations", ylab = "Huber loss")
```




```
#gd1$xmat[, 151:200]
```

The algorithm now odesn't converge but oscillates. We can deduce that the coefficients are changing periodically which is further suported by checking the xmat from gd1 above.

7.

```
# compare the estimated and true coefficients
gd$x
```

```
##  [1] -0.87346579  0.61828938  0.87989797 -0.04910821  0.07277491
##  [6]  0.10229815 -0.12513246 -0.14559243 -0.11903666 -0.02250130
```

```
sparse.grad.descent <- function(f, x0, max.iter = 200, step.size = 0.05, stopping.deriv = 0.01, ...) {

  n    <- length(x0)
  xmat <- matrix(0, nrow = n, ncol = max.iter)
  xmat[,1] <- x0

  for (k in 2:max.iter) {
    # Calculate the gradient
    grad.cur <- grad(f, xmat[ ,k-1], ...)
```

```
    # Should we stop?
    if (all(abs(grad.cur) < stopping.deriv)) {
      k <- k-1; break
    }

    # Move in the opposite direction of the grad
    tmp <- xmat[ ,k-1] - step.size * grad.cur
    tmp <- ifelse(abs(tmp) <= 0.05, 0, tmp)
    xmat[ ,k] <- tmp
  }

  xmat <- xmat[ ,1:k] # Trim
  return(list(x = xmat[,k], xmat = xmat, k = k))
}

# sparse estimates
gd.sparse <- sparse.grad.descent(huber.loss, x0 = rep(0, p), step.size = 0.001, stopping.deriv = 0.1)
# final estimates
gd.sparse$x
```

```
##  [1] -0.8944804  0.6332991  0.8823860  0.0000000  0.0000000  0.0000000
##  [7]  0.0000000  0.0000000  0.0000000  0.0000000
```

8.

```
lm_coe <- as.numeric(lm(y~x-1)$coef)
gd_coe <- gd$x
spa_gd_coe <- gd.sparse$x

mean((lm_coe-b)^2)
```

```
## [1] 0.1186581
```

```
mean((gd_coe-b)^2)
```

```
## [1] 0.01208955
```

```
mean((spa_gd_coe-b)^2)
```

```
## [1] 0.005610471
```

Not surprisingly, sparse gradient descent estimate has the least mean squared error and is thus the best.

9.

```
set.seed(10)
y <- x %*% b + rt(n, df=2)

gd_new <- grad.descent(huber.loss, x0 = rep(0, p), step.size = 0.001, stopping.deriv = 0.1)
gd_new$x
```

```
##  [1] -0.46329748  0.92390614  0.92287242 -0.06526259  0.24633002
##  [6] -0.04406371  0.01858892 -0.18921630  0.19479185 -0.18395820
```

```r
mean((gd_new$x - b)^2)
```

```
## [1] 0.02869228
```

```r
spa_gd_new <- sparse.grad.descent(huber.loss, x0 = rep(0, p), step.size = 0.001, stopping.deriv = 0.1)
spa_gd_new$x
```

```
##  [1] 0.0000000 0.7850744 0.9398727 0.0000000 0.0000000 0.0000000 0.0000000
##  [8] 0.0000000 0.0000000 0.0000000
```

```r
mean((spa_gd_new$x - b)^2)
```

```
## [1] 0.0500853
```

The new coefficients from gradient descent looks nothing new. But that from sparse gradient descent is a little different since we notice that the first element in the coefficient is 0 which is actually not 0. As for the MSE, the gradient descent is superior and we may deduce the sparse gradient descnt is more variable.

10.

```r
mse_gd <- c()
mse_sgd <- c()

for(i in 1:10) {
  y <- x %*% b + rt(n, df=2)

  gd_new <- grad.descent(huber.loss, x0 = rep(0, p), step.size = 0.001, stopping.deriv = 0.1)
  mse_gd <- c(mse_gd, mean((gd_new$x - b)^2))

  spa_gd_new <- sparse.grad.descent(huber.loss, x0 = rep(0, p), step.size = 0.001, stopping.deriv = 0.1)
  mse_sgd <- c(mse_sgd, mean((spa_gd_new$x - b)^2))
}

mean(mse_gd)
```

```
## [1] 0.02495459
```

```r
mean(mse_sgd)
```

```
## [1] 0.02650818
```

```r
min(mse_gd)
```

```
## [1] 0.01430856
```

```r
min(mse_sgd)
```

```
## [1] 0.0006265157
```

Strictly speaking, the average MSE from the gradient descent is lower. As for the minimum MSE, the sparse gradient descent is significantly less which is in line with the former result that sparse gradient descent is more variable.