James-Stein Estimator: a visualization

Luis F. Campos

December 12, 2017

Problem Setup

$$X \sim N((\mu, \mu, \mu), I_3)$$

We consider three estimators of the vector (μ, μ, μ) for this visualization:

- Mean: X
- James-Stein: $JS = (1 - \frac{k-2}{||X||^2})X$
- James-Stein Positive Part: $JS_+ = (1 - \frac{k-2}{||X||^2})_+ X$

These renderings are an implementation of visualizations seen on Naftali Harris' blog.

```
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
# Install these packages
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
library('sphereplot')
library('scatterplot3d')
library('car')


cols = c("#8aa5d1", "#e84a7c", 1)


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
# positive part funciton for use in JS+
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
pos = function(x){
    out = x
    for(i in 1:length(x)){
        out[i] = max(0, x[i])
    }
    out
}

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
# Makes several 3-dimensional renderings to show what JS transformation
#   is doing. We consider level sets of a 3-dimensional Normal centered
#   at (1,1,1) and we consider applying JS as well as JS+ to those points.
#   We then measure the fraction of those points helped by the transformation
#   and print a percentage. We then plot those points helped by the
#   transformaiton.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
plot_JS = function(rad){
    points = sph2car(expand.grid(long = seq(0, 360, 5), lat = seq(-90, 90, 5)),
```

```r
    radius = rad, deg = TRUE)
points = points + matrix(1, nrow = nrow(points), ncol = ncol(points))
points = rbind(c(1, 1, 1), points)

# Apply James-Stein Transformation to the set of points
JS.points = matrix((1 - 1/apply(points, 1, function(x){sum(x^2)})),
    nrow = nrow(points), ncol = ncol(points)) * points

# Apply James-Stein Transformation (positive part) to the set of points
JS_pos.points = matrix(pos(1 - 1/apply(points, 1, function(x){sum(x^2)})),
    nrow = nrow(points), ncol = ncol(points)) * points

# merge points for plotting
all.points = rbind(c(1, 1, 1), points, JS.points)
all.points2 = rbind(c(1, 1, 1), points, JS_pos.points)

# label points for point colors
lab = c(rep(1, nrow(points)), rep(2, nrow(points)))

# Plot level set
print(paste('level set with radius', rad))
scatter3d(points[,1],points[,2],points[,3],
    point.col = c(1, rep("#8aa5d1", nrow(points))),
    surface=FALSE, xlab = 'X[1]', ylab = 'X[2]',
    zlab = 'X[3]', pch = 19, cex.symbols = 0.3)
pause = readline()
print(paste('JS applied to level set with radius', rad))
# plot level set (black) with JS transformation applied (red)
scatter3d(all.points[,1],all.points[,2],all.points[,3],
    point.col = c(cols[3], cols[lab]), surface=FALSE,
    xlab = 'X[1]', ylab = 'X[2]', zlab = 'X[3]',
    xlim = c(0, 5), pch = 19, cex.symbols = 0.3,
    main = 'James-Stein Estimator')
pause = readline()
# plot level set (black) with JS+ transformation applied (red)
# print(paste('JS+ applied to level set with radius', rad))
# scatter3d(all.points2[,1],all.points2[,2],all.points2[,3],
#   point.col = c(cols[3], cols[lab]), surface=FALSE,
#   xlab = 'X[1]', ylab = 'X[2]', zlab = 'X[3]',
#   xlim = c(0, 5), pch = 19, cex.symbols = 0.3,
#   main = 'James-Stein Estimator')

d1 = as.matrix(dist(all.points))[1,-1]

# Fraction of points whose original values are closer to (1,1,1) than the JS values
JS_helps = d1[1:nrow(points)] > d1[-c(1:nrow(points))]
print(paste('Fraction of points "helped" by JS shrinkage: ',
    mean(JS_helps)))
pause = readline()

print('Points "helped" by JS shrinkage')
points_plot = points[which(JS_helps),]
scatter3d(points_plot[,1],points_plot[,2],points_plot[,3],
```

```r
        surface=FALSE, xlab = 'X[1]', ylab = 'X[2]', zlab = 'X[3]',
        xlim = c(0, 5), point.col = "#8aa5d1", pch = 19, cex.symbols = 0.3,
        main = 'James-Stein Estimator')


}

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
# Usage: Given a radius (defining a level set), this funciton plots
#   several 3-d renderings of the level set points, and points
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
plot_JS(1)
plot_JS(0.2)
plot_JS(1.5)
plot_JS(2)




# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
# This function analyzes data from multivariate Normal
# to understand performance of JS and JS+ shrinkage
#
# 1. Randomly generate points from a multivariate Normal centered
#    at (mu,...,mu) of dimension k
# 2. Applies JS and JS+ to those points
# 3. Calculates distance to (mu,...,mu) for each of the three
#    transformations
# 4. Compares distances to (mu,...,mu) as well as average distance.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
library('mvtnorm')

dist_shrink = function(mu, k, rep = 1000){
    mu.vec = rep(mu, k)
    points = rmvnorm(rep, mu.vec)

    JS.points = matrix((1 - (k-2)/apply(points, 1, function(x){sum(x^2)})),
        nrow = nrow(points), ncol = ncol(points)) * points
    JS_pos.points = matrix(pos(1 - (k-2)/apply(points, 1, function(x){sum(x^2)})),
        nrow = nrow(points), ncol = ncol(points)) * points

    d1 = as.matrix(dist(rbind(mu.vec, points)))[-1,1]
    d2 = as.matrix(dist(rbind(mu.vec, JS.points)))[-1,1]
    d3 = as.matrix(dist(rbind(mu.vec, JS_pos.points)))[-1,1]

    h1 = density(d1)
    h2 = density(d2)
    h3 = density(d3)

    plot(h1, lwd = 2, ylim = c(0, max(c(h1$y, h2$y, h3$y))),
        main = 'Euclidian Distance to (mu,...,mu)')
    lines(h2, col = 2, lwd = 2)
    lines(h3, col = 3, lwd = 2)
```

```
    abline(v = mean(d1), lwd = 2)
    abline(v = mean(d2), col = 2, lwd = 2)
    abline(v = mean(d3), col = 3, lwd = 2)

    legend('topright', c('Y', 'JS', 'JS+',
        paste('mu: ', mu,' and k: ', k, sep = '')),
    col = c(1, 2, 3, 0), lwd = 2)
}


dist_shrink(1, 3)

for(mu in 0:10) {
    dist_shrink(mu, 3)
    readline()
}


for(dim in 3:20) {
    dist_shrink(1, dim)
    readline()
}
```