Minimal Empirical Density Estimation

Matthew Leonawicz

December 24, 2014

1 Use case 1: Climate

Temperature and precipition data from SNAP's downscaled climate models are often used to make inferences about future trends and uncertainty among potential climate scenarios. This generally is done by looking at specific statistics culled from the model outputs, usually the mean value over some combination of factors of interest such as seasonal period or geographical region. But what about when we want to look at an entire distribution of data at once? Furthermore, we want to visualize many distributions in quick succession, distributions of temperature or precipitation from different months, seasons, years, decades, locations, climate models, scenarios, etc.

This is a case where I extract a relatively coarse density estimate a priori. Subsequently, this small, stored estimate, is what is actually used to regenerate an accurate simulation of the original model output. This means no inefficient, slow lugging around of big data. Compress it down, release it in similar to original form later.

The following is used in the `AR4_AR5_extract.R` script which is part of the SNAP data QA/QC project and feeds into the Shiny app for comparing CMIP3 and CMIP5 downscaled climate model outputs. There is a function for estimating densities. In this case I wanted to avoid any `NA` values and for precipitation I wanted to ensure densities did not include positive probability over any interval containing negative values. The only other variable being analyzed was temperature. Things like this are important to code for in nuanced ways when the goal is to apply such a function to a large number of datasets. Effort must go into dealing with rare idiosyncrasies in the data and their effects.

```
denFun <- function(x, n, variable) {
    x <- x[!is.na(x)]
    dif <- diff(range(x))
    z <- density(x, adjust = 2, n = n, from = min(x) - 0.05 * dif, to = max(x) +
        0.05 * dif)
    if (variable == "pr" && any(z$x < 0))
        z <- density(x, adjust = 2, n = n, from = 0, to = max(x) + 0.05 * dif)
    as.numeric(c(z$x, z$y))
}
```

On the other end, in this case tucked within the aforementioned Shiny app, the function, `density2bootstrap` is used to simulate new draws from the estimated density. This is much faster and more efficient than trying to load an enormous data set. In this app, the `ggplo2` graphics rely on the sample for plotting which is why the bootstrapping occurs following loading of the density estimate, and why there is no subsequent code for fitting a new density estimate to the bootstrap sample.

```
density2bootstrap <- function(d, n.density, n.boot = 10000, interp = FALSE,
    n.interp = 1000, ...) {
    n.fact <- n.boot/n.density
    n.grp <- nrow(d)/n.density
    d$Index <- rep(1:n.grp, each = n.density)
    d2 <- data.frame(lapply(d, rep, n.fact), stringsAsFactors = FALSE)
```

```
    prob.col <- which(names(d2) %in% c("Prob", "Index"))
    d2 <- d2[order(d2$Index), -prob.col]
    d2$Val <- as.numeric(vapply(X = 1:n.grp, FUN = function(i, d, n, interp,
        n.interp, ...) {
        p <- list(x = d$Val[d$Index == i], y = d$Prob[d$Index == i])
        if (interp) p <- approx(p$x, p$y, n = n.interp)
        round(sample(p$x, n, prob = p$y, rep = T), ...)
    }, FUN.VALUE = numeric(n.boot), d = d, n = n.boot, interp = interp, n.interp = n.interp,
        ...))
    d2
}
```

However, for a complete picture of how these functions work together, it is important to see the documentation for the projects of which they are a part.