A TUTORIAL FOR INVIDIOUS COMPARISONS: AN R VINAIGRETTE

ROGER KOENKER

Abstract. These notes are intended to be a rough introductory guide to the software for the paper: Invidious Comparisons: Ranking and Selection as Compound Decisions.

1. Introduction

I will try to illustrate all of this with our data on U.S. dialysis centers. A prerequisite for using our methods is the nonparametric maximum likelihood estimator for Gaussian mixtures incorporated in our R package **REBayes**. It, in turn, relies on the installation of the R package **Rmosek** and **mosek**, the Danish convex optimization language. Details about installation and free academic licensing is available from `https://docs.mosek.com/9.2/rmosek/install-interface.html`.

The data for this exercise can be accessed with the command,

```
library(REBayes)
## Loading required package:  Matrix
load("D.Rda")
str(D)
##  num [1:3230, 1:14, 1:4] 1.084 0.724 0.805 1.005 0.769 ...
##  - attr(*, "dimnames")=List of 3
##   ..$ : chr [1:3230] "102501" "102502" "102503" "102504" ...
##   ..$ : chr [1:14] "2004" "2005" "2006" "2007" ...
##   ..$ : chr [1:4] "Y" "Ylo" "Yup" "ED"
```

The ever-useful `str` command reveals that the `D` object produced by the `load` command is an array consisting of 14 years of data on each of 3230 centers on four variables. The `Y` component is observed mortality, `Yup` and `Ylo` are officially reported upper and lower bounds on predicted mortality and `ED` is expected mortality all taken from a Cox model that purports to adjust for patient mix in the 3230 centers. As descibed in the paper we convert this Poisson varsion of the data into Gaussian form in the first few lines of the function `fit1d`

```
source("fit1d.R")
fit1d
## function (subset, D, u = NULL, bwt = 2, rtol = 1e-12, ...)
## {
##     y = sqrt(D[, subset, 1])
##     d = D[, subset, 1] * D[, subset, 4]
```

December 14, 2021. A genre manifesto for R Vinaigrettes is available at `http://davoidofmeaning.blogspot.com/2016/12/r-vinaigrettes.html`.

```
##     e = D[, subset, 4]
##     s = 1/sqrt(4 * D[, subset, 4])
##     w = 1/s^2
##     id = rep(1:nrow(y), each = length(subset))
##     y = as.vector(t(y))
##     w = as.vector(t(w))
##     d = as.vector(t(d))
##     e = as.vector(t(e))
##     W = tapply(w, id, sum)
##     D = round(tapply(d, id, sum))
##     E = round(tapply(e, id, sum))
##     S = tapply(w * y, id, sum)/W
##     M = tapply(y, id, length)
##     V = (tapply(w * y^2, id, sum) - W * S^2)/(M - 1)
##     if (!length(u))
##         u = 300
##     f = GLmix(S, sigma = 1/sqrt(W), u = u, rtol = rtol, ...)
##     fs = KWsmooth(f, bw = bwKW(f, bwt))
##     fp = Pmix(D, exposure = E)
##     list(f = f, fs = fs, fp = fp, y = y, id = id, w = w, W = W,
##         D = D, E = E, S = S, M = M, V = V)
## }
```

In the last few lines of this function we use this Gaussian version of the data to estimate a mixing distribution structure that is called `f` and another called `fs` which smooths `f`. A Poisson mixture model is also estimated for purposes of later comparison called `fp`, which I won't consider further here. In addition, various intermediate outputs are attached in the returned object that come in handy in the selection process as described below.

Next we need to use what has been just computed to do selection subject to our capacity constraint α and the FDR constraint γ. This looks quite nasty because we were interested in quite a few different decision rules. There are two basic selection functions, naturally called `selectL1d` and `selectR1d`. They are called to to construct decision boundaries of various types in the graphics function `level_plot` that is used to produce our figures illustrating conflict and agreement among the decision rules. Let's consider just selecting the left tail observations subject to fixed α and γ constraints and ignore the plotting functionality.

The function `selectL1d` is rather ugly due to the fact that it is computing selections for several rules all together. If we focus on only the tail probability rules, `TPKW` and `TP` and ignore what we refer to as the naive rules and the posterior mean rules, then the fog clears a little bit. The function returns a list consisting of two objects: `A` is a Boolean array indicating the units selected by the rule that imposes both capacity and FDR control, and `B` indicates units selected when only capacity control is imposed. A typical invocation of the function might look like this:

```
z <- fit1d(1:3,D)
sL = selectL1d(z, alpha = 0.22, gamma = 0.2)
str(sL)
## List of 2
##  $ A: num [1:3230, 1:8] 0 0 0 0 0 0 0 0 0 0 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:8] "TPKW" "TP" "PMKW" "PMKWs" ...
##  $ B: num [1:3230, 1:8] 0 0 0 0 1 0 0 0 0 0 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:8] "TPKW" "TP" "PMKW" "PMKWs" ...
apply(sL$A,2,sum)
##  TPKW    TP  PMKW PMKWs  MLE1 P-val   E&M    JS
##    29   230    29   229   152   225   322   322
```

Here we use only the first 3 years of the data to produce an object `z` which is then fed into the `selectL` function with specified constraint parameters to produce selections for the various rules. Again, the `str` command reveals that the arrays `A` and `B` are 3230 by 6 arrays that contain indicators for whether each center is selected into the left tail by the corresponding column rule. The `apply` command computes the number of selected centers for each of the six rules using the capacity and FDR control constraints. In this instance FDR control imposes a rather severe limit on the number of centers selected.