When does a Poisson log-normal look like a negative binomial?

A. J. Rominger

7/20/2018

We want the expected value and variance of zero-truncated negative binomial maximum log likelihoods given N draws from a Poisson log-normal distribution with parameters μ and σ.

Maybe we can use the underlying math of the `logLikZ` function to do this. In `logLikZ` we have the following:

```
# hypothetical distribution of probabilities
p0 <- .p0(x)

# hypothetical mean and var
m <- sum(p0 * exp(p0)) * n
v <- sum((m/n - p0)^2 * exp(p0)) * n

.p0 <- function(x) {
    n0 <- 1:10^5
    dfun <- getdfun(x)

    p0 <- dfun(n0, log = TRUE)
    p0 <- p0[is.finite(p0)]

    if(exp(p0[length(p0)]) > .Machine$double.eps^0.75) {
        n0add <- (n0[length(p0)] + 1):10^6
        p0add <- dfun(n0add, log = TRUE)
        p0add <- p0add[is.finite(p0add)]

        p0 <- c(p0, p0add)
    }

    return(p0)
}
```

Taking for example the term for the mean `m <- sum(p0 * exp(p0)) * n`, we see that the `p0` term is the vector of all possible log likelihood values while the `exp(p0)` term is the probability of observing any one of those log likelihood values. So if we want to know the mean and variance of likelihoods of, e.g., a negative binomial model given negative binomial data, we leave these expressions as is. If, on the other hand, we want the likelihoods of a negative binomial model given Poisson lognormal data we would need `exp(q0)` instead of `exp(p0)` where `q0` comes from the Poisson lognormal distribution.

Let's make a function for this idea, that we can calculate the mean and variance of a log likelihood function where the model and data can either derive from the same distribution or different. What's more we can take the difference of the two resulting distributions of likelihoods, one where model and data come from the same distribution, and the other where they don't.

```
# we'll be using SADs from pika so might as well load now
library(pika)

#' @param d1 a function describing the PMF of the first distribution
#' @param d2 function for the second distribution
```

```r
#' @param N number of observations (i.e. species)
#' @details note: this function assumes the data from from the first
#' distribution `d1`
llDist <- function(d1, d2, N) {
    p <- makeP(d1)
    q <- makeP(d2)

    mPgivenP <- sum(p * exp(p)) * N
    vPgivenP <- sum((mPgivenP / N - p)^2 * exp(p)) * N

    # p and q might be different lengths, so standardize them before
    # combining them
    xMax <- min(length(p), length(q))
    p <- p[1:xMax]
    q <- q[1:xMax]

    mQgivenP <- sum(q * exp(p)) * N
    vQgivenP <- sum((mQgivenP / N - q)^2 * exp(p)) * N

    mPQDiff <- mPgivenP - mQgivenP
    vPQDiff <- vPgivenP + vQgivenP

    return(c(mean = mPQDiff, sd = sqrt(vPQDiff)))
}

# helper function to make a vector of log probabilites
makeP <- function(dfun) {
    n0 <- 1:10^5

    p0 <- dfun(n0, log = TRUE)
    p0 <- p0[is.finite(p0)]

    if(exp(p0[length(p0)]) > .Machine$double.eps^0.75) {
        n0add <- (n0[length(p0)] + 1):10^6
        p0add <- dfun(n0add, log = TRUE)
        p0add <- p0add[is.finite(p0add)]

        p0 <- c(p0, p0add)
    }

    return(p0)
}

# test it out
llDist(d1 = function(n, ...) dplnorm(n, 1, 1, ...),
       d2 = function(n, ...) dtnegb(n, 1, 1, ...),
       N = 100)
```

```
##     mean       sd
## 114.3782  46.4207
```

Note `d1` and `d2` need to be functions that already contain the desired parameterization of the probability distributions they represent. So the next thing we need is to be able to find the maximum likelihood parameterization of the negative binomial model for an arbitrary data set generated with a known parameterization

of the Poisson log-normal. We can do that by minimizing the KL divergence between the two:

$$D_{NB|PNL} = \sum_{i=1}^{\infty} P_{PLN}(n_i) \log\left(\frac{P_{PLN}(n_i)}{P_{NB}(n_i)}\right)$$

Rearranging a little can help us write a more efficient optimization routine:

$$D_{NB|PNL} = \sum_{i=1}^{\infty} P_{PLN}(n_i)\left(\log P_{PLN}(n_i)\right) - \sum_{i=1}^{\infty} P_{PLN}(n_i)\left(\log P_{NB}(n_i)\right)$$

We are seeking the optimal parameters for the negative binomial, so the terms involving only PLN need only be computed once in the optimaztion

```
#' @param d1 is the Poisson log-normal PMF, consistent with \code{d1}
#' in \code{lldist}

minKL_pln_nb <- function(plnParam) {
    p <- makeP(function(n, ...) {
        dplnorm(n, plnParam[1], plnParam[2], ...)
    })
    n0 <- 1:length(p)

    plnEnt <- sum(exp(p) * p)

    KL_pln_nb <- function(param) {
        if(any(param < .Machine$double.eps^0.75)) {
            return(NA)
        } else {
            q <- dtnegb(n0, param[1], param[2], log = TRUE)
            return(plnEnt - sum(exp(p) * q, na.rm = TRUE))
        }
    }

    o <- optim(plnParam, KL_pln_nb)

    o <- c(plnParam, o$par, o$value, o$convergence)
    names(o) <- c('pln_mu', 'pln_sig', 'tnb_mu', 'tnb_k', 'kl_dist', 'converg')
    return(o)
}

# test it out
foo <- minKL_pln_nb(c(2, 1))
foo
```

```
##      pln_mu     pln_sig      tnb_mu       tnb_k     kl_dist     converg
##  2.00000000  1.00000000 11.17001018  0.77815489  0.02790705  0.00000000
```

Now we can look at the likelihood distance for the optimal parameters

```
dist1 <- llDist(d1 = function(n, ...) dplnorm(n, 2, 1, ...),
                d2 = function(n, ...) dtnegb(n, foo['tnb_mu'], foo['tnb_k'], ...),
                N = 100)
dist1
```

```
##      mean        sd
```

```
##  2.790705 17.608859
```

Finally we can use this normally distributed distance to calculate the probability of failing to distinguish between PLN and NB:

```
pnorm(2, dist1[1], dist1[2])
```

```
## [1] 0.482092
```

So we're about 50/50 as to whether we reject NB in favor of PLN, i.e. there's about a 0.5 probability that the two likelihoods (and thus their AIC, sence number of parameters are equal) are within 2 units of each other.

We can put this all together and look at when we reject NB across the parameter space of the PLN:

```
plnPars <- expand.grid(seq(0.1, 10, length.out = 10), seq(0.01, 5, length.out = 10))

llInfo <- lapply(1:nrow(plnPars), function(i) {
    pln <- unlist(plnPars[i, ])
    o <- minKL_pln_nb(pln)
    thisDist <- llDist(d1 = function(n, ...) dplnorm(n, pln[1], pln[2], ...),
                       d2 = function(n, ...) dtnegb(n, o['tnb_mu'], o['tnb_k'], ...),
                       N = 100)
    o <- c(o, thisDist, pVal = pnorm(2, thisDist[1], thisDist[2]))

    return(o)
})

llInfo <- do.call(rbind, llInfo)
head(llInfo)
```

```
##      pln_mu pln_sig     tnb_mu      tnb_k       kl_dist converg
## [1,]    0.1    0.01   1.105167  2838.6763 -2.052787e-07       0
## [2,]    1.2    0.01   3.319521   575.5827  5.679249e-06       0
## [3,]    2.3    0.01   9.974350 10863.3289 -1.399525e-07       0
## [4,]    3.4    0.01  29.966065  9501.4578 -1.372466e-07       0
## [5,]    4.5    0.01  90.024506  9833.7814 -9.588199e-08       0
## [6,]    5.6    0.01 270.440682  9948.5598 -1.412499e-07       0
##                mean        sd      pVal
## [1,] -2.052787e-05 10.969001 0.5723396
## [2,]  5.679249e-04  8.865750 0.5892139
## [3,] -1.399525e-05  9.873853 0.5802592
## [4,] -1.372466e-05  9.970038 0.5794953
## [5,] -9.588199e-06  9.989925 0.5793390
## [6,] -1.412499e-05  9.996478 0.5792878
```

Plotting it:

```
library(socorro)
library(viridis)
```

```
## Loading required package: viridisLite
```

```
layout(matrix(1:2, nrow = 1), widths = c(2, 1))

par(mar = c(3, 3, 0, 1) + 0.5, mgp = c(2.5, 0.75, 0))
image(unique(llInfo[, 'pln_sig']), unique(llInfo[, 'pln_mu']),
      matrix(llInfo[, 'pVal'], nrow = length(unique(llInfo[, 'pln_mu']))),
      col = viridis(30),
      xlab = 'PLN mu', ylab = 'PLN sigma')
```

```
par(mar = c(3, 0, 0, 0) + 0.5)
plot(simpECDF(llInfo[, 'pVal']), col = quantCol(sort(unique(llInfo[, 'pVal'])),
                                                viridis(30)),
     xlab = 'Likelihood divergence', yaxt = 'n', pch = 16)
abline(v = 0.05)
```