Advanced analysis using baySeq; generic distribution definitions

Thomas J. Hardcastle

October 17, 2014

1 Generic Prior Distributions

baySeq now offers complete user-specification of underlying distributions. This vignette describes using baySeq under this protocol. Familiarity with standard (negative-binomial) baySeq is assumed; please consult the other vignettes for a description of this approach.

Analysis is carried out through specification of a `densityFunction` class. The primary value in a `densityFunction` object is the `@density` slot, a user-defined function that should take variables `x`, `observables` and `parameters`. `x` corresponds to a row of data in a `countData` object. `observables` is a list object of observed values that may influence the model. By default, the `@libsizes` and `@seglens` values of the `countData` object will be internally appended to this list, unless objects with these names are otherwise specified by the user. `parameters` is a list object of parameters to be empirically estimated from the data with the `getPriors` function and used to estimate likelihoods with the `getLikelihoods` function. The `@dist` function should return a vector of log-likelihood values (or NA for invalid parameter choices) of the same length as the input variable `x`.

Other required slots of the `densityFunction` object are `initiatingValues`, a vector of initiating values to be used in optimisation of the parameters to be used in the `@dist` slot (and thus defining the length of the parameter object) and `equalOverReplicates`, a specification of which parameters are fixed for every replicate group and which may vary for different experimental conditions. If only one parameter is variable over experimental conditions, the Nelder-Mead optimisation used may be unstable, and one-dimensional optimisation with user defined functionally specified lower and upper bounds may (optionally) be provided; otherwise, Nelder-Mead will be attempted.

Optionally a function may be provided in `@stratifyFunction` to stratify the data and improve prior estimation in the tails where the `samplesize` argument in the getPriors function is less than the row dimension of the `countData` object. If this is provided, the `@stratifyBreaks` slot should give the number of strata to be used.

Below a model is constructed based on the normal distribution. The standard deviation is assumed to be constant for a given row of data across all experimental condidtions, while the means (normalised by library scaling factor) are allowed to vary across experimental conditions.

If parallelisation is available, it is useful to use it.

```
> if(require("parallel")) cl <- makeCluster(8) else cl <- NULL
```

```
> library(baySeq)
> normDensityFunction <- function(x, observables, parameters) {
+   if(any(sapply(parameters, function(x) any(x < 0)))) return(rep(NA, length(x)))
+   dnorm(x, mean = parameters[[2]] * observables$libsizes, sd = parameters[[1]], log = TRUE)
+ }
> normDensity <- new("densityFunction", density = normDensityFunction, initiatingValues = c(0.1, 1),
+                 equalOverReplicates = c(TRUE, FALSE),
+                 lower = function(x) 0, upper = function(x) 1 + max(x) * 2,
+                 stratifyFunction = rowMeans, stratifyBreaks = 10)
```

We construct the `countData` object as before.

```
> data(simData)
> CD <- new("countData", data = simData,
+           replicates = c("simA", "simA", "simA", "simA", "simA",
+             "simB", "simB", "simB", "simB", "simB"),
```

```
+           groups = list(NDE = c(1,1,1,1,1,1,1,1,1,1),
+                         DE = c(1,1,1,1,1,2,2,2,2,2))
+           )
> libsizes(CD) <- getLibsizes(CD)
> densityFunction(CD) <- normDensity
```

We can then fit priors and calculate posterior likelihoods based on our specified distributional model. The distributional model is specified in the 'getPriors' function and will be automatically used in the 'getLikelihoods' function

```
> normCD <- getPriors(CD, cl = cl)
> normCD <- getLikelihoods(normCD, cl = cl)
```

.

Similarly, we can construct a generic version of the negative-binomial model.

```
> nbinomDensityFunction <- function(x, observables, parameters) {
+   if(any(sapply(parameters, function(x) any(x < 0)))) return(NA)
+   dnbinom(x, mu = parameters[[1]] * observables$libsizes * observables$seglens, size = 1 / parameters
+ }
> densityFunction(CD) <- new("densityFunction", density = nbinomDensityFunction, initiatingValues = c(0
+                            equalOverReplicates = c(FALSE, TRUE),
+                            lower = function(x) 0, upper = function(x) 1 + max(x) * 2,
+                            stratifyFunction = rowMeans, stratifyBreaks = 10)
> nbCD <- getPriors(CD, cl = cl)
> nbCD <- getLikelihoods(nbCD, cl = cl)
```

.

We can compare this to the standard analysis of these data.

```
> CD <- getPriors.NB(CD, cl = cl)
> CD <- getLikelihoods(CD, cl = cl)
```

.

The generic negative-binomial data performs almost identically to standard baySeq. The methods differ in that the standard baySeq uses quasi-maximum-likelihood to estimate the priors, while generic baySeq uses maximum-likelihood (since no generic method exists for quasi-maximum-likelihood on arbitrary distributions).

2 Different Model Priors

It is now possible to use different model priors for different subsets of the countData object. If we expect a certain class of genes (for example) to have a different prior likelihood towards differential expression than another such class, we can separate the two sets and estimate (or set) the model priors independently.

Let us suppose that we have reason to believe that the first hundred genes in the 'CD' object are likely to behave differently to the remaining genes. Then

```
> CDv <- getLikelihoods(nbCD, modelPriorSets = list(A = 1:100, B = 101:1000), cl = cl)
```

.

The model priors used are recorded in the @priorModels slot.

```
> CDv@priorModels
```

```
$A
NDE  DE
0.2 0.8

$B
 NDE   DE
0.97 0.03
```

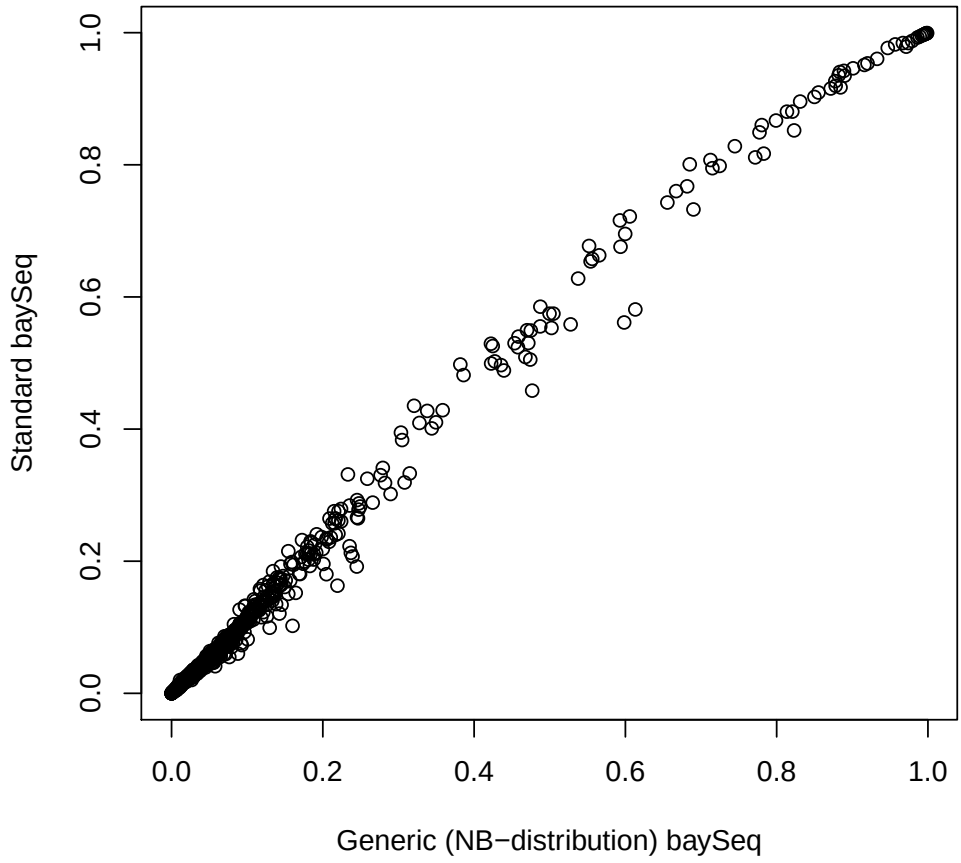

Figure 1: Likelihoods of DE estimated by standard/generic baySeq"

We can see the difference in performance by computing the ROC curves as before. Using different model priors can substantially improve performance, although obviously we have cheated here by splitting exactly those data simulated as DE and those as none-DE. It should also be recognised that this approach may bias downstream analyses; e.g. GO enrichment analysis.

Several pre-existing distributions are built into baySeq. Here we use a pre-developed zero-inflated negative binomial distribution to analyse zero-inflated data.

```
> data(zimData)
> CD <- new("countData", data = zimData,
+           replicates = c("simA", "simA", "simA", "simA", "simA",
+             "simB", "simB", "simB", "simB", "simB"),
+           groups = list(NDE = c(1,1,1,1,1,1,1,1,1,1),
+                         DE = c(1,1,1,1,1,2,2,2,2,2))
+           )
> libsizes(CD) <- getLibsizes(CD)
> densityFunction(CD) <- nbinomDensity
> CD <- getPriors(CD, cl = cl)
> CD <- getLikelihoods(CD, cl = cl)

.

> CDz <- CD
> densityFunction(CDz) <- ZINBDensity
```

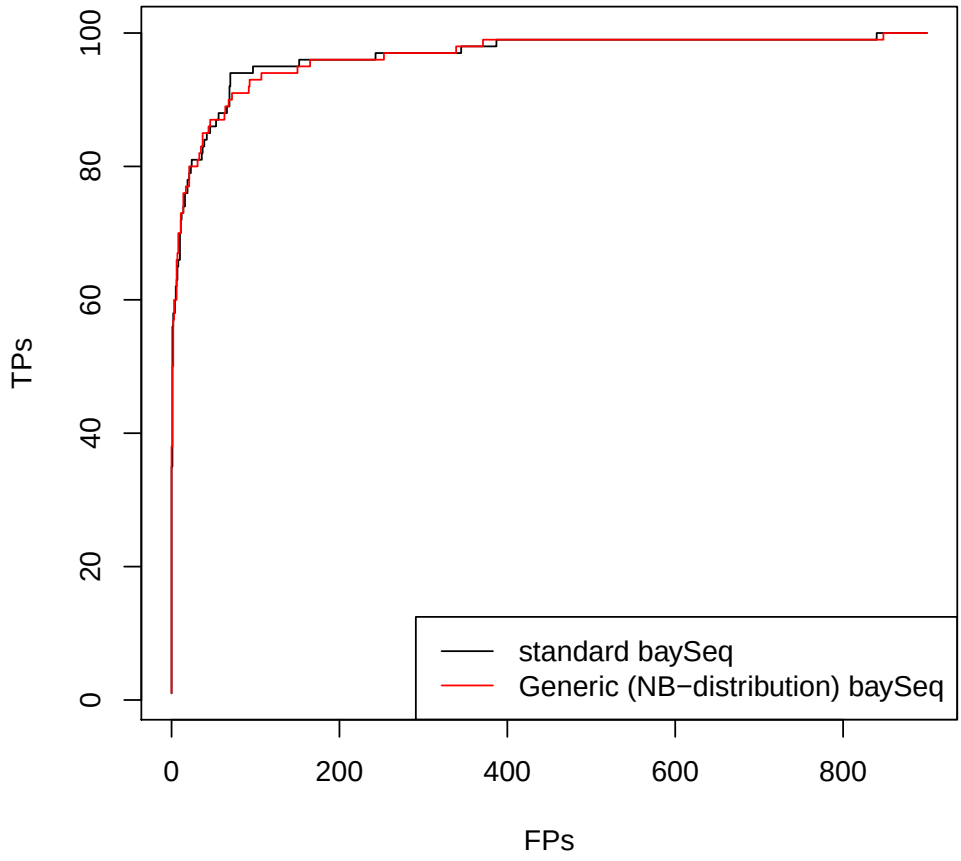

Figure 2: ROC curves estimated by standard/generic baySeq"

```
> CDz <- getPriors(CDz, cl = cl)
> CDz <- getLikelihoods(CDz, cl = cl)
```

.

Finally, we shut down the cluster (assuming it was started to begin with).

```
> if(!is.null(cl)) stopCluster(cl)
```

Session Info

```
> sessionInfo()
```

```
R version 3.1.1 Patched (2014-09-25 r66681)
Platform: x86_64-apple-darwin13.1.0 (64-bit)

locale:
[1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] stats4    parallel  stats     graphics  grDevices utils     datasets  methods
[9] base
```

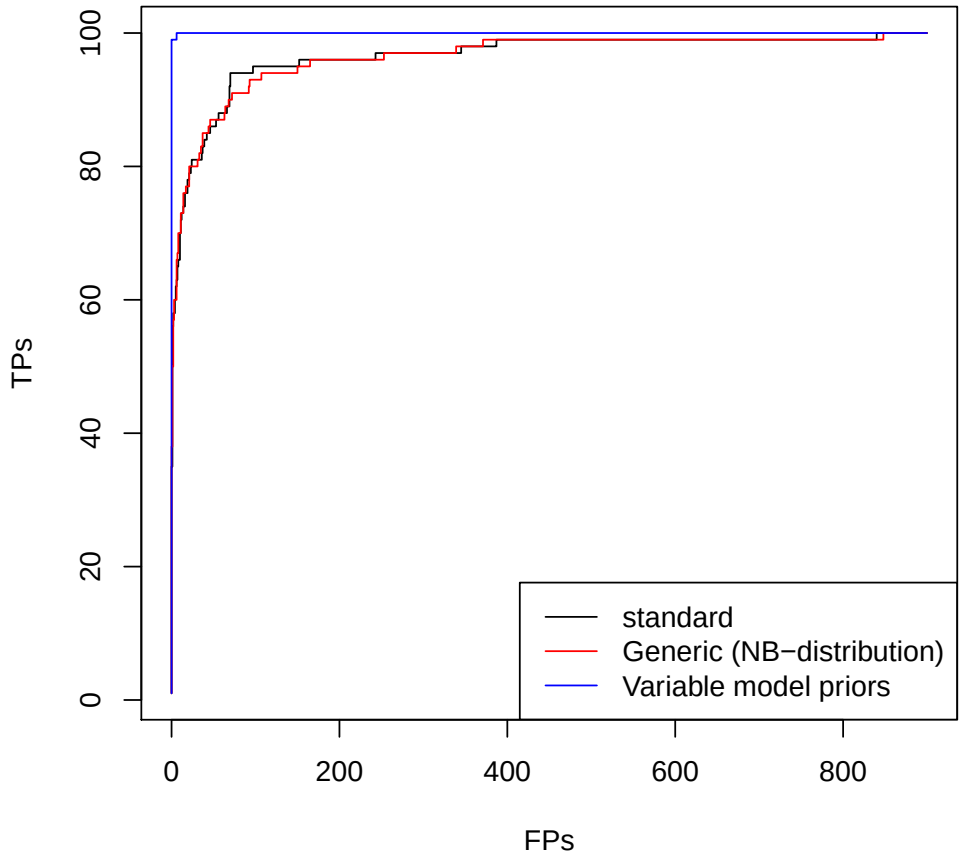

Figure 3: ROC curves estimated by standard/generic/variable model priors baySeq"

```
other attached packages:
[1] baySeq_2.0.50        abind_1.4-0          GenomicRanges_1.18.1 GenomeInfoDb_1.2.0
[5] IRanges_2.0.0        S4Vectors_0.4.0      BiocGenerics_0.12.0

loaded via a namespace (and not attached):
[1] BiocStyle_1.4.1 XVector_0.6.0   tools_3.1.1
```