CNVRG vignette

Joshua Harrison

2020-09-08

'CNVRG' wraps functionality provided by 'RStan' and 'Stan' in a simple to use interface (Stan Development Team 2018). Credit should be given to the 'Stan' team if 'CNVRG' is used. Additionally, please cite the 'CNVRG' package and associated Molecular Ecology Resources paper (Harrison et al. 2020a). Thanks!

'CNVRG' facilitates Dirichlet multinomial modelling of relative abundance data, such as those generated via sequencing of microbiomes. We envision the software being of particular use for microbial and community ecologists. The variational inference algorithm provided by 'Stan' extends the utility of the model to modern datasets with many thousands of parameters to estimate.

In this vignette, we demonstrate how to use 'CNVRG' on a simple dataset. We urge new users to carefully examine the files used. Data must be formatted properly for 'CNVRG' to work!

We also recommend that application of 'CNVRG' to larger data (e.g., a matrix of several thousand by several hundred) be performed remotely as modelling can take some compute time. As a head's up to new users, it is worth subsetting one's data and ensuring code correctness and model completion prior to analyzing a large dataset.

Data to be processed by 'CNVRG' should look like the following matrix, with a treatment field followed by fields containing count data. Note that count data must be integers and must be read by 'R' as such (not factors or strings). The treatment field must be encoded as a factor or string. Use str() to determine the format of your data. See 'DataCamp', 'swirl' or other 'R' tutorials if you do not know how to change the encoded format for your data.

```
fungi[1:3,1:5]
##      treatment Otu4 Otu58 Otu77 Otu86
## 10 treated_neg  816     4     3     0
## 12 treated_neg  277     2     0     0
## 23 treated_neg  479     0     0     0
```

Before modelling it is important to ensure that the data are organized such that all replicates from a treatment group are grouped together. For instance, all "treatment1" replicates should be followed by all "treatment2" replicates in the matrix in row wise fashion. Replicates from different treatment groups should not be interdigitated. This is because the hierarchical nature of the model shares information among replicates within a treatment group. If replicates are jumbled up then information will not be shared properly and model output will be incorrect.

For instance, consider:

```
fungi$treatment
##   [1] treated_neg    treated_neg    treated_neg    treated_neg
##   [5] treated_neg    treated_neg    treated_neg    treated_neg
##   [9] treated_neg    treated_neg    treated_neg    treated_neg
##  [13] treated_neg    treated_neg    treated_neg    treated_neg
##  [17] treated_neg    treated_neg    treated_neg    treated_neg
##  [21] treated_neg    treated_neg    treated_neg    treated_neg
##  [25] treated_neg    treated_neg    treated_neg    treated_neg
```

```
##  [29] treated_neg    treated_neg    treated_neg    treated_neg
##  [33] treated_neg    treated_neg    treated_neg    treated_neg
##  [37] treated_neg    treated_neg    treated_plus   treated_plus
##  [41] treated_plus   treated_plus   treated_plus   treated_plus
##  [45] treated_plus   treated_plus   treated_plus   treated_plus
##  [49] treated_plus   treated_plus   treated_plus   treated_plus
##  [53] treated_plus   treated_plus   treated_plus   treated_plus
##  [57] treated_plus   treated_plus   treated_plus   treated_plus
##  [61] treated_plus   treated_plus   treated_plus   treated_plus
##  [65] treated_plus   treated_plus   treated_plus   treated_plus
##  [69] treated_plus   treated_plus   treated_plus   treated_plus
##  [73] treated_plus   treated_plus   treated_plus   untreated_neg
##  [77] untreated_neg  untreated_neg  untreated_neg  untreated_neg
##  [81] untreated_neg  untreated_neg  untreated_neg  untreated_neg
##  [85] untreated_neg  untreated_neg  untreated_neg  untreated_neg
##  [89] untreated_neg  untreated_neg  untreated_neg  untreated_neg
##  [93] untreated_neg  untreated_neg  untreated_neg  untreated_neg
##  [97] untreated_neg  untreated_neg  untreated_neg  untreated_neg
## [101] untreated_neg  untreated_neg  untreated_neg  untreated_neg
## [105] untreated_neg  untreated_neg  untreated_neg  untreated_neg
## [109] untreated_neg  untreated_neg  untreated_neg  untreated_neg
## [113] untreated_plus untreated_plus untreated_plus untreated_plus
## [117] untreated_plus untreated_plus untreated_plus untreated_plus
## [121] untreated_plus untreated_plus untreated_plus untreated_plus
## [125] untreated_plus untreated_plus untreated_plus untreated_plus
## [129] untreated_plus untreated_plus untreated_plus untreated_plus
## [133] untreated_plus untreated_plus untreated_plus untreated_plus
## [137] untreated_plus untreated_plus untreated_plus untreated_plus
## [141] untreated_plus untreated_plus untreated_plus untreated_plus
## [145] untreated_plus untreated_plus untreated_plus untreated_plus
## Levels: treated_neg treated_plus untreated_neg untreated_plus
```

All replicates for each of the four treatment groups in these example data are placed together in the matrix. The first and last indices for replicates in each of these treatment groups are used to tell the function which rows in the matrix correspond to a particular sampling group. See the vectors passed in as arguments for 'starts' and 'ends' in the example immediately below.

Note if zeros exist in the data, then a pseudocount should be added. A one is used in this case.

```
fungi[,2:length(fungi)] <- 1 + fungi[,2:length(fungi)]
modelOut <- varHMC(countData = fungi,
       starts = c(1,39,76,113),
       ends = c(38,75,112,148),
       chains = 2,
       burn = 500,
       samples = 2000,
       thinning_rate = 2,
       cores = 1,
       params_to_save = c("pi", "p"))
```

Incidentally, if desired, information can be shared among multiple treatment groups through rearranging the data and specifying new start and end indices. For example, here we share information among replicates within a treatment group. We specify the indices for the first replicate in each sampling group via the vector provided as an argument to 'starts'. Similarly, the indices for the last replicate in each treatment group are fed in a as a vector to 'ends'. We have four elements in each of these vectors because there are four treatment

groups. If instead, we wished to combine the first two treatment groups and share information accordingly, then the start and end vectors could be modifed to be 'starts = c(1,76,113)' and 'ends = c(75,112,148)' respectively.

After running the model it is important to check convergence statistics. One way to do this is with summary (from the RStan package).

```
head(summary(modelOut, pars = "pi", probs =c(0.025, 0.975))$summary)
##                mean      se_mean           sd        2.5%       97.5%
## pi[1,1] 0.926310573 1.268586e-04 0.0034110463 0.919962382 0.933129071
## pi[1,2] 0.007153096 2.633784e-05 0.0008501852 0.005581631 0.008779607
## pi[1,3] 0.003731700 1.605914e-05 0.0005529292 0.002708087 0.004878408
## pi[1,4] 0.002863655 1.378174e-05 0.0004666183 0.002015567 0.003839107
## pi[1,5] 0.002717714 1.342067e-05 0.0004664906 0.001925412 0.003731347
## pi[1,6] 0.002628350 1.286881e-05 0.0004375578 0.001857257 0.003574774
##             n_eff      Rhat
## pi[1,1]  722.9951 0.9988115
## pi[1,2] 1041.9979 1.0005305
## pi[1,3] 1185.4802 0.9992176
## pi[1,4] 1146.3451 0.9997364
## pi[1,5] 1208.1962 1.0021317
## pi[1,6] 1156.0967 0.9991340
```

Rhat scores should be near 1. It is advisable to check other measures of model performance (number of effective samples, Geweke's statistic, trace plots) The 'shinystan' package also provides excellent visualizations of model performance. The package can be called like this: 'shinystan::launch_shinystan(modelOut)'. If you are unfamiliar with convergence diagnostics then please see the 'Stan' documentation.

At the moment, diagnostic tools are not well developed for variational inference.

Using estimates of relative abundances

If model diagnostics seem sufficient then analyses can be performed using relative abundance estimates. Use the 'extract' function from 'RStan' to assign samples for parameters of interest to an object. These samples can then be passed to downstream analyses.

While we advocate using samples describing posterior distributions for parameters of interest whenever possible, we acknowledge that sometimes it is convenient to obtain point estimates for those parameters. To do so for pi parameters we can use the 'CNVRG' function 'extract_point_estimate' like so:

```
point_est <- extract_point_estimate(modelOut = modelOut, countData = fungi, treatments = 2)
```

Differential relative abundance testing

One common analysis is to compare relative abundances of features between treatment groups. This is commonly referred to as 'differential abundance testing' in the microbial ecology and functional genomics literatures. 'CNVRG' provides functions to ease this analysis. Simply pass in the extracted pi parameters (the function does not work for other parameters) and specify the count data used. The count data must follow the exact format as those modelled and be in the same order.

```
diff_abund_test <- diff_abund(model_output = modelOut, countData = fungi)
```

This function subtracts the posterior distribution of the pi paramater for each feature in one treatment group from the pi parameter distribution in other treatment groups. The function outputs a matrix of proportions

that describes the proportion of the distribution of differences that is greater than zero, for each comparison. In this example, the comparison between treatment 1 and treatment 3 provided the following results.

```
diff_abund_test[3,]
##                   comparison  Otu4     Otu58     Otu77 Otu86  Otu9
## 3 treatment_1_vs_treatment_3 0.998 0.2906667 0.3206667 0.168 0.156
##       Otu11 Otu10     Otu54 Otu42     Otu40     Otu74     Otu12     Otu72
## 3 0.1433333 0.172 0.1486667  0.14 0.1526667 0.1353333 0.1226667 0.1533333
##       Otu79     Otu96 Otu94      Otu6     Otu92     Otu70     Otu71 Otu97
## 3 0.1433333 0.1606667 0.144 0.3306667 0.1266667 0.2606667 0.1466667  0.15
##       Otu62  Otu7     Otu99     Otu76    Otu100
## 3 0.1493333 0.156 0.1526667 0.1526667 0.1253333
```

This means that for Otu 4 nearly 100% of the samples obtained after subtracting the pi distributions for that Otu from treatment 1 and treatment 3 were greater than zero. This means that the there is high certainty that the pi value for Otu 4 in treatment 1 was greater than in treatment 3. Stated another way, this means that the relative abundance of Otu 4 was greater in treatment 1 than in treatment 3.

However, only approximately a third of samples for the distribution of differences for Otu 58 were above zero. So we have much less certainty that this Otu differed in relative abundance between treatment group 1 and treatment group 3.

Because the 'diff_abund' function provides insight into the proportion of samples ABOVE zero, values that are very large and very small (e.g., >0.95 or <0.05) denote a high certainty of an effect of treatment group.

Diversity calculation

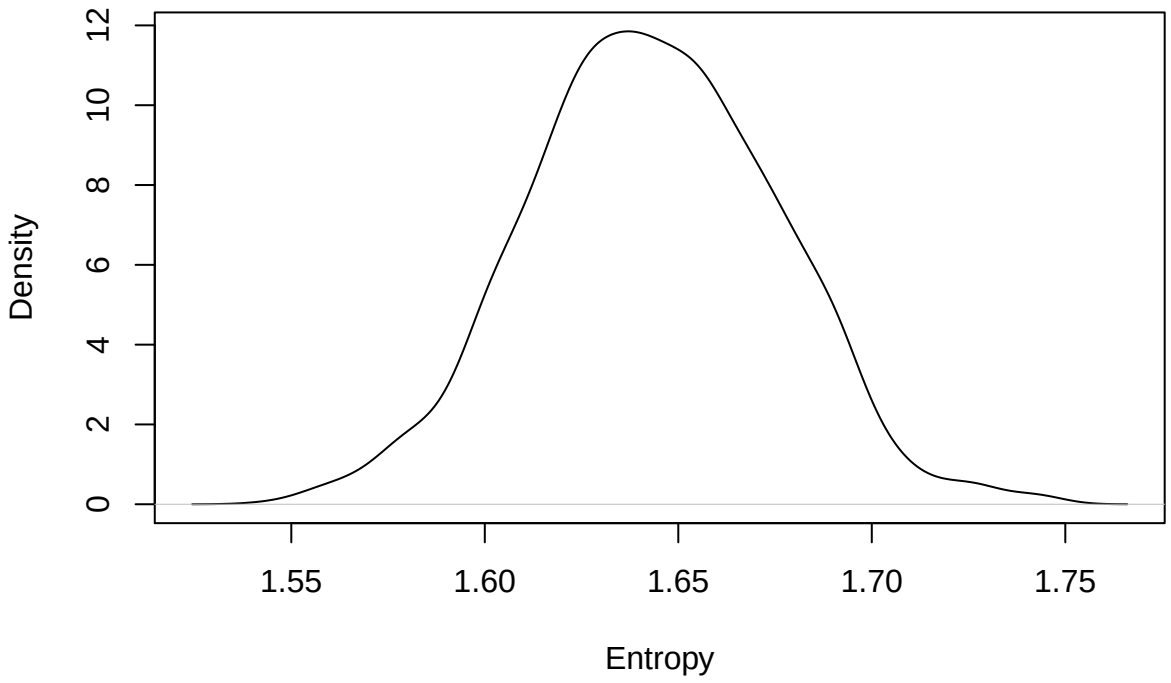
Often some measure of the information content of data is desirable. This can be calculated using diversity entropies, such as the Shannon index. Ecologists use these metrics all the time, though they can be useful in many other fields as well. 'CNVRG' allows propagation of uncertainty in relative abundance estimates through diversity entropy calculations.

To calculate diversity entropies use the 'diversity_calc' function:

```
entropies <- diversity_calc(model_output = modelOut, countData = fungi,
                            entropy_measure = 'shannon',equivalents = T)
```

We do not provide explicit plotting functions for entropy posteriors as we suspect users will have formatting desires that preclude the use of canned functions. However, a simple way to plot entropy posteriors is via density plots using base 'R'. For example:

```
plot(density(entropies[[1]][[1]]),
     xlab = "Entropy",
     ylab = "Density",
     main = "")
```



Transforming relative abundance data to absolute abundance estimates

If an internal standard (ISD) has been used, then it is a simple matter to transform samples of posterior distributions so that relative abundance estimates are provided as ratios with the internal standard's relative abundance as the denominator. This transforms the data so that each field is represented in relation to the internal standard. Since the standard should represent the same starting absolute abundance, this transformation accounts for the problems of compositionality inherent to relative abundance data, to some extent at least. There are many situations where the standard may fail and we direct users to Harrison et al. 2020b for a description of these situations. Still, we suggest that an internal standard provides important benefits for many studies.

We have provided a simple function in 'CNVRG' called 'isd_transform' that facilitates the aforementioned transformation. Users must specify which field corresponds to the ISD. For the sake of example, let's say that the second index corresponds with the ISD.

```
transformed_data <- isd_transform(model_output = modelOut, countData = fungi, isd_index = 2)
```

It is worth checking that the correct index for the ISD was provided. Examine the output and make sure that the field for the index is filled with ones and that field is indeed the ISD. Remember when determining the appropriate index to account for the fact that the original data had sample names in field one (this field should not be counted when determining an appropriate index). Transformed data can be used in the 'diversity_calc', 'diff_abund', and 'extract_point_estimates' functions.

Literature cited

Harrison, J. G., Calder, W. J., Shastry, V., & Buerkle, C. A. (2020a). Dirichlet multinomial modelling outperforms alternatives for analysis of microbiome and other ecological count data. Molecular Ecology Resources, 20(2)

Harrison, J., Calder, W. J., Shuman, B. N., & Buerkle, C. A. (2020b). The quest for absolute abundance: the use of internal standards for DNA based community ecology. Molecular Ecology Resources (Accepted as of Aug. 2020)

Stan Development Team (2018). RStan: the R interface to Stan. R package version 2.18.2.